Metody připojování periferií

BI-MPP

Přednáška 1

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

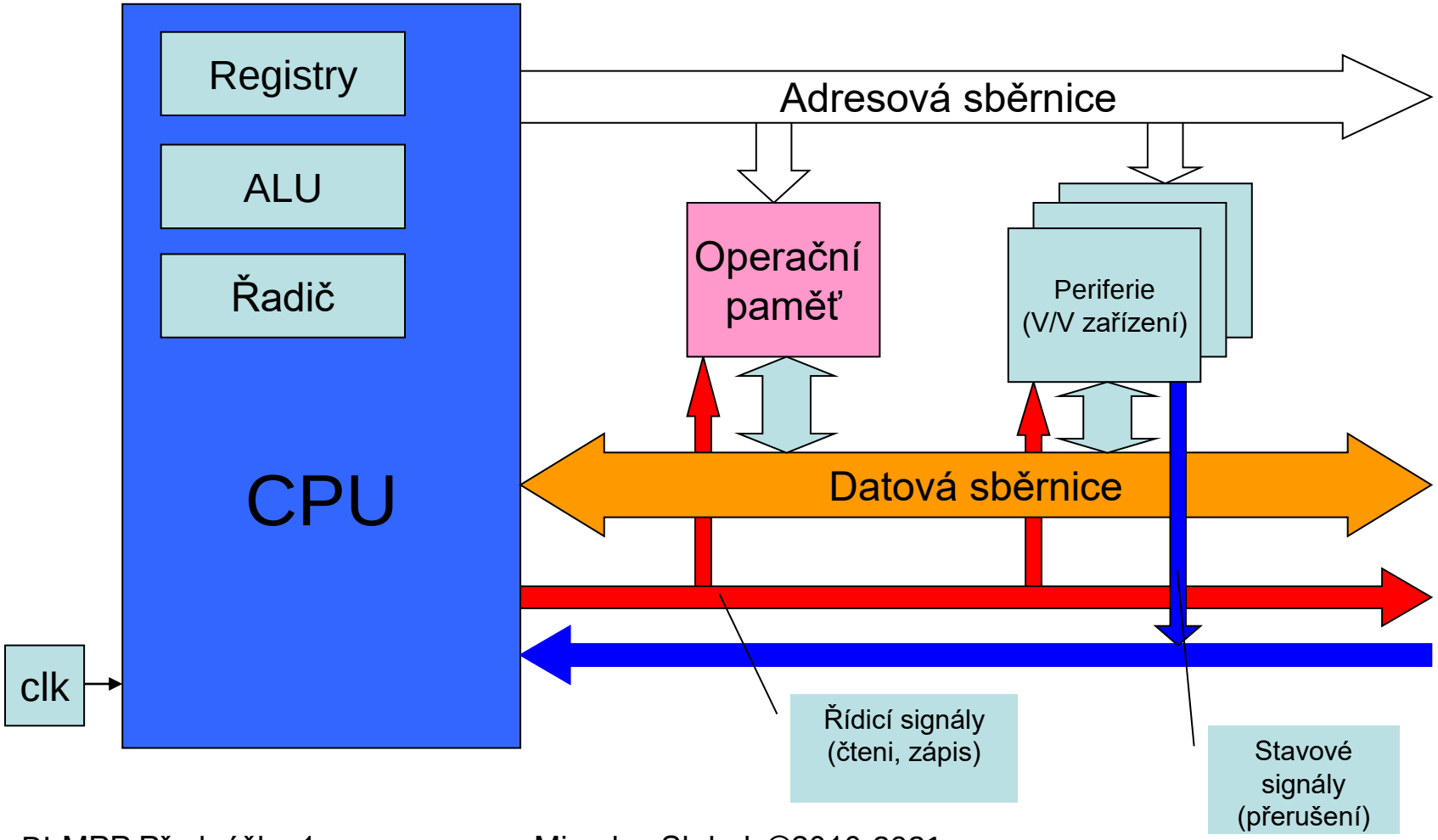
Agenda

- Úvod
- Architektura PC
- Sběrnice

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- PCI Express™ Base Specification Revision 1.0a. PCI Sig. April 15, 2003

Blokové schéma počítače (odpovídá procesoru 8086)



Nejstarší motherboard PC XT (procesor 8086)

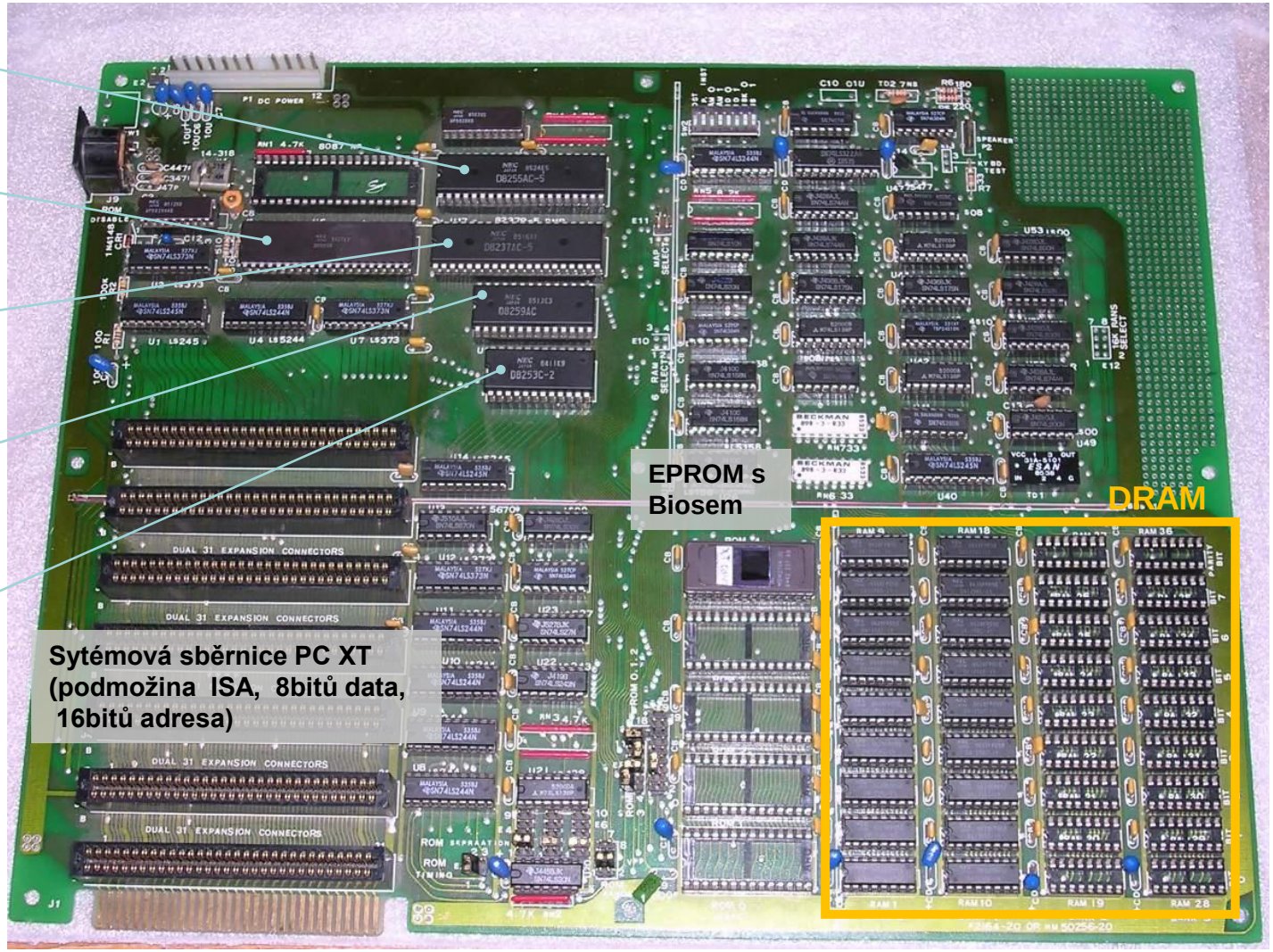
Paralelní port (8255)

Procesor 8086

Řadič DMA 8237

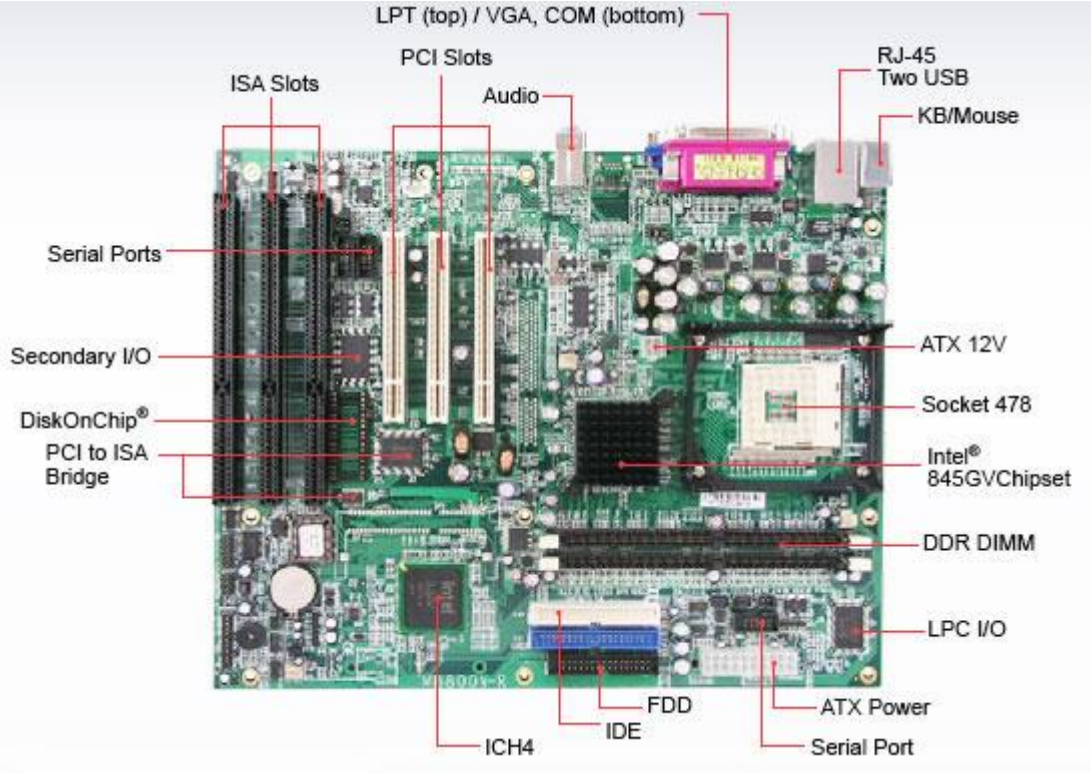
Řadič přerušení (8259)

Čítač/časovač (8253)



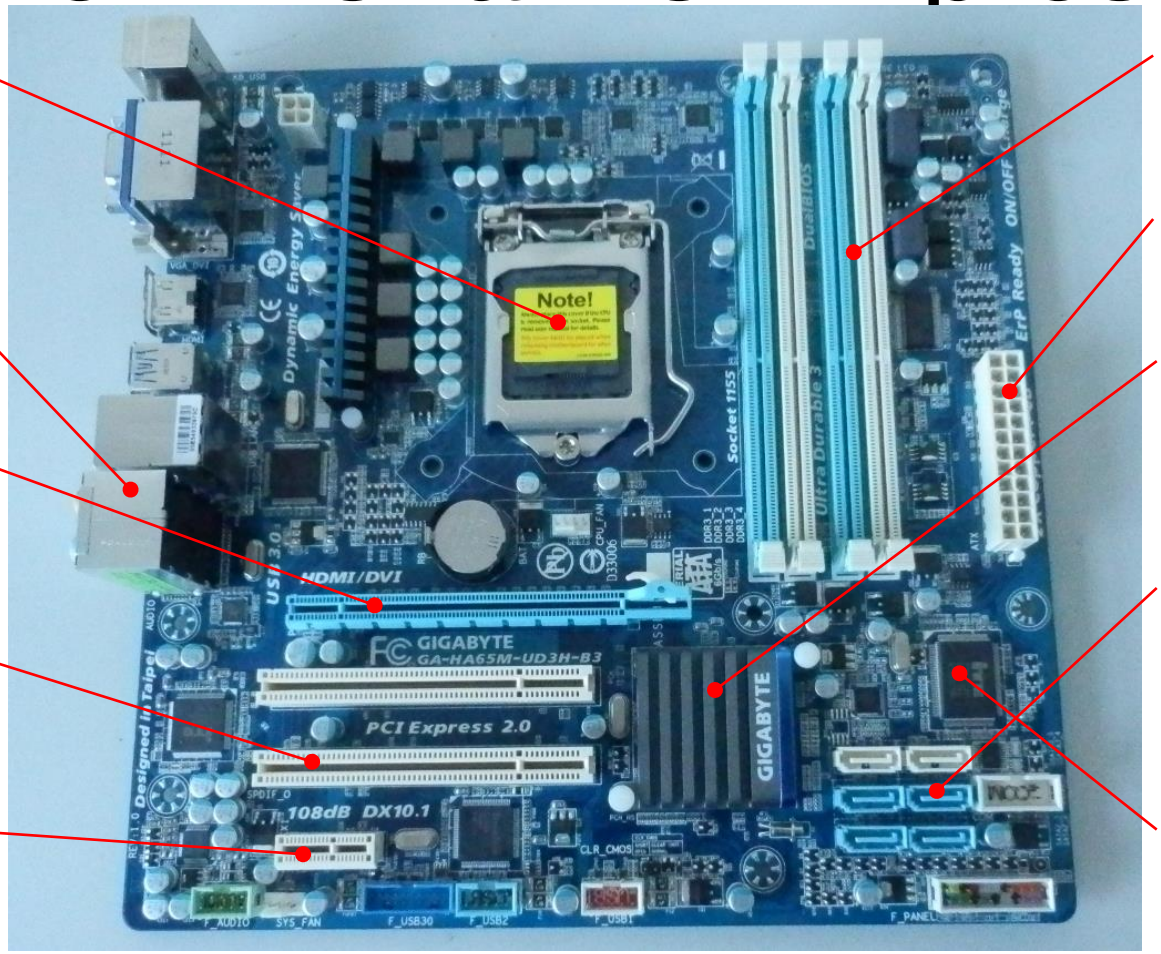
Zdroj: http://oldcomputer.info/log/items/2018/201810270142041.jpg

Starší motherboard (přechod mezi ISA a PCI)



Zdroj: https://www.rampcsystems.com/3-isa-slot-motherboard-detailed-specifications

Starší motherboard (přechod mezi PCI a PCI Express)



Patice pro procesor

Audio konektory

PCI Express 16x

Konektory PCI

PCI Express 1x

Konektory pro paměťové moduly

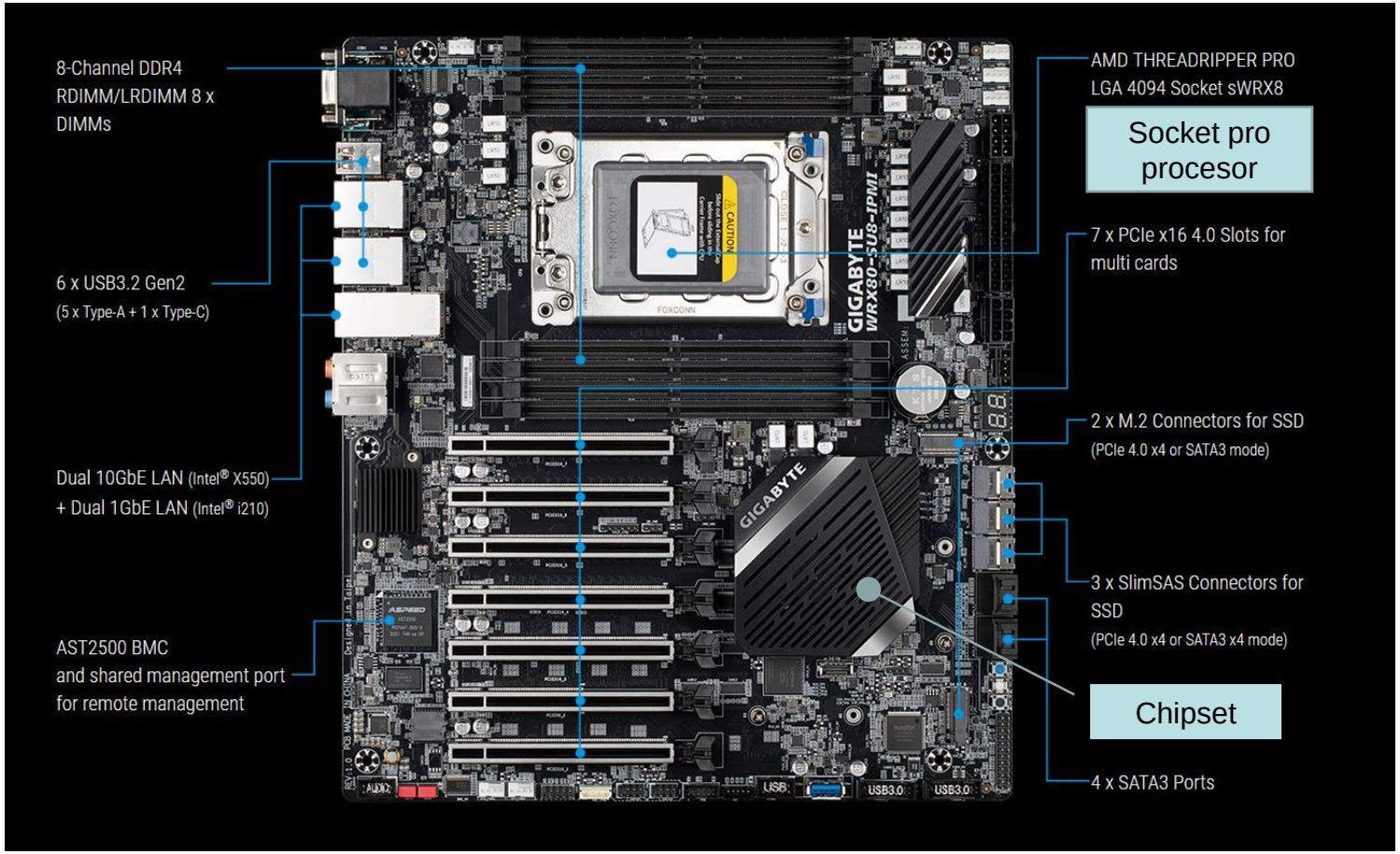
Napájecí konektor ATX

Jižní můstek

Konektory SATA

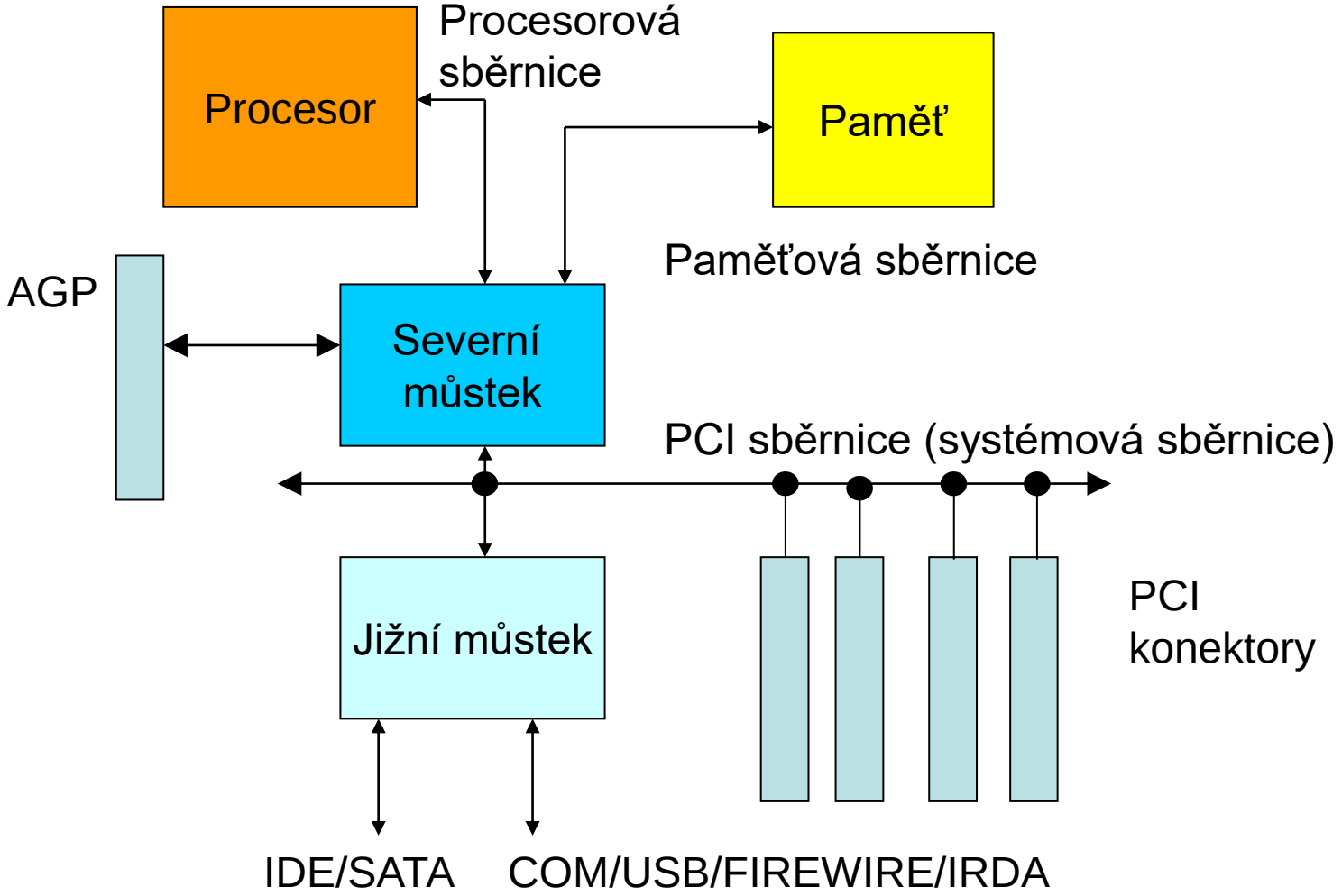
ITE Chip COM, PS2, KB, Myš

Moderní motherboard (pouze PCI Expres)



Zdroj: https://www.gigabyte.com/Motherboard/WRX80-SU8-IPMI-rev-10#kf

Architektura PC (starší architektura)



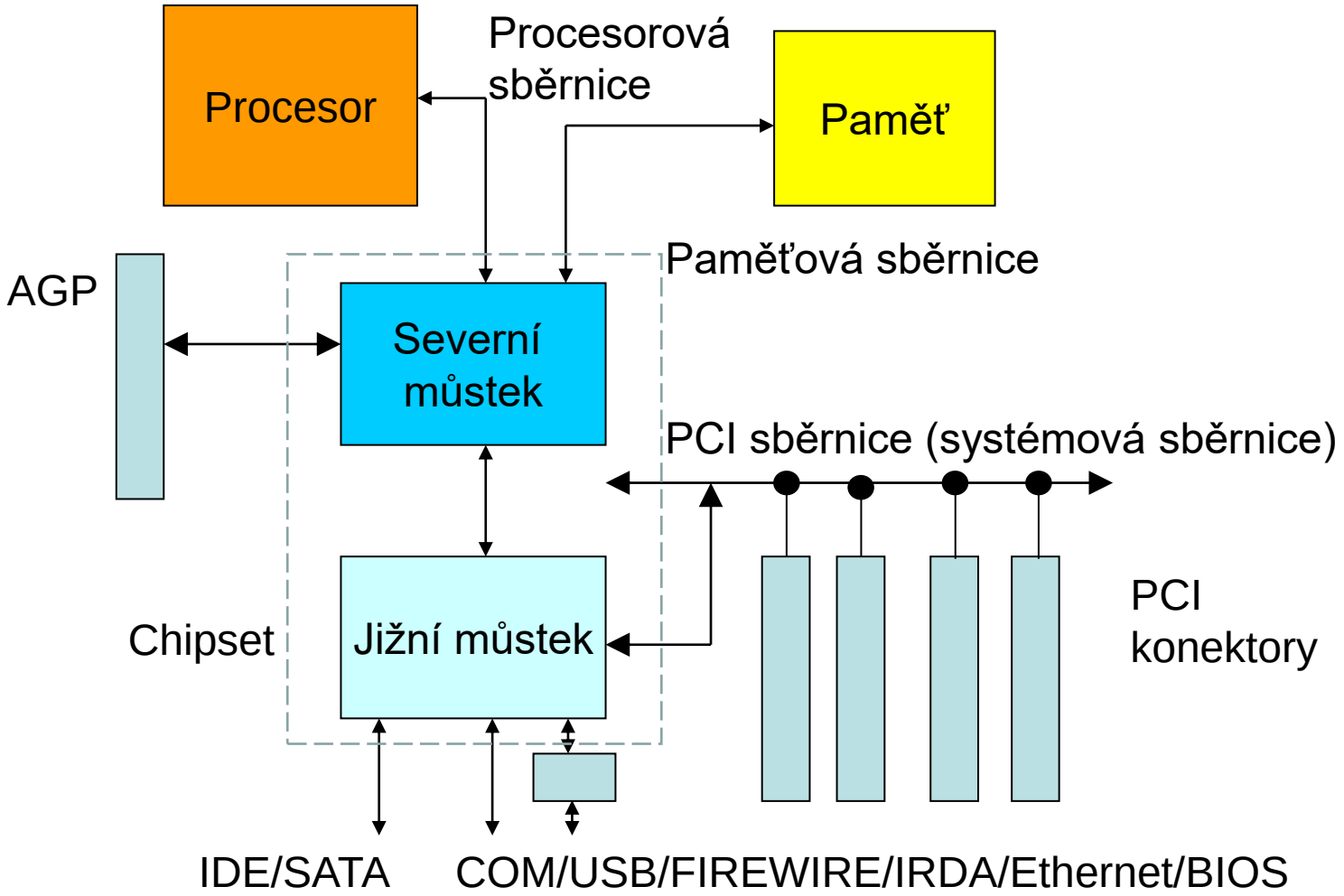
Severní můstek

- Převádí procesorovou sběrnici procesoru na PCI nebo PCI Expres
- Dříve obsahoval řadič paměti, nyní je řadič paměti přímo u procesoru
- V současnosti v řadě systémů severní můstek nenajdeme, některé funkcionality přebírá jižní můstek

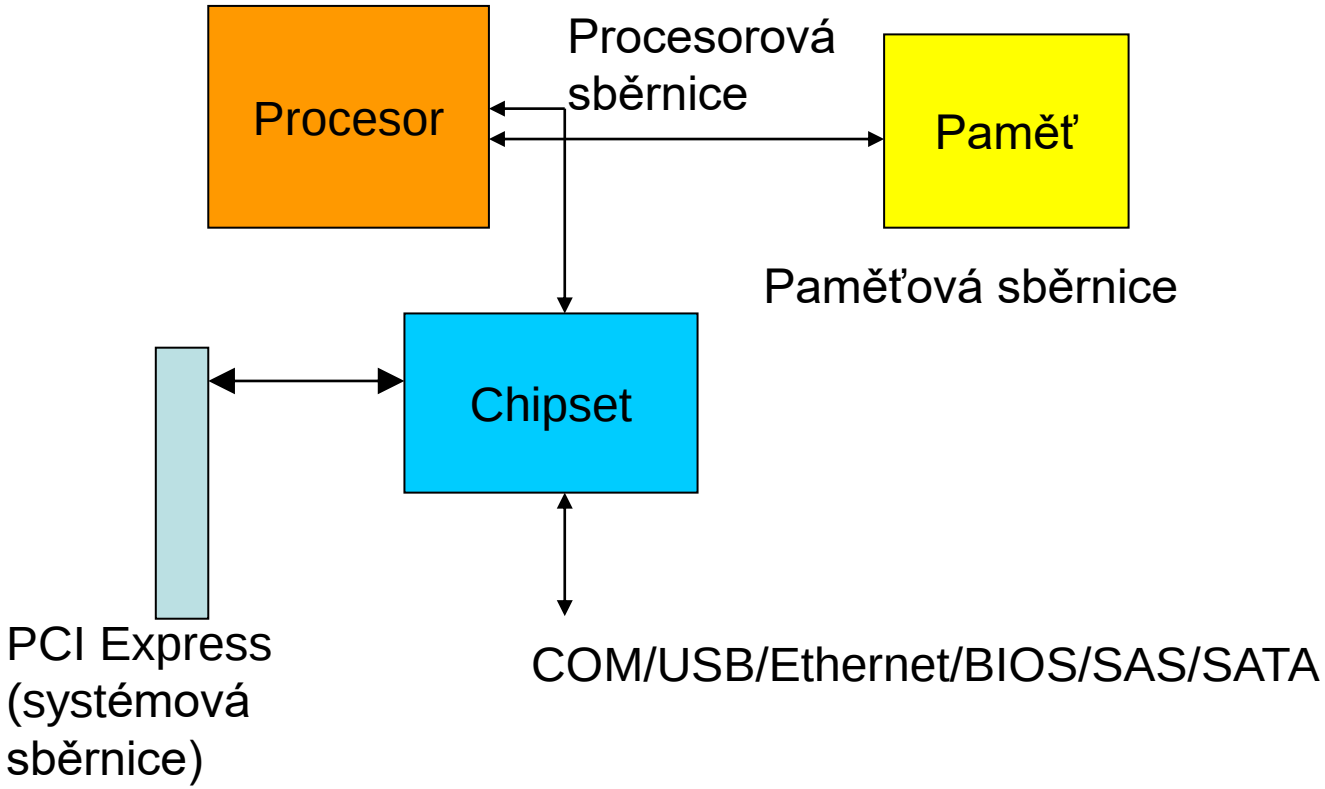
Jižní můstek

- Obsahuje řadiče periferních obvodů
  - Řadič USB
  - Řadič FIREWIRE (zmizel)
  - Sériové porty (téměř zmizely)
  - Paralelní port (zmizel)
  - Síťový řadič (Ethernet)
- Převádí PCI na jiné typy sběrnic
  - SATA
  - IDE (zmizelo)
  - PCMCIA (zmizelo)
  - USB
  - SAS

Architektura PC (severní/jižní můstek)



Architektura PC nejmodernější



Sběrnice

- Soustava vodičů, které propojují obvody počítače a jsou určeny pro přenos dat, adres, řídících a stavových signálů.
- Přenos dat se řídí buď proprietárním nebo standardizovaným protokolem.
- Časování (průběhy signálů) a způsob sychronizace určují maximální propustnost sběrnice

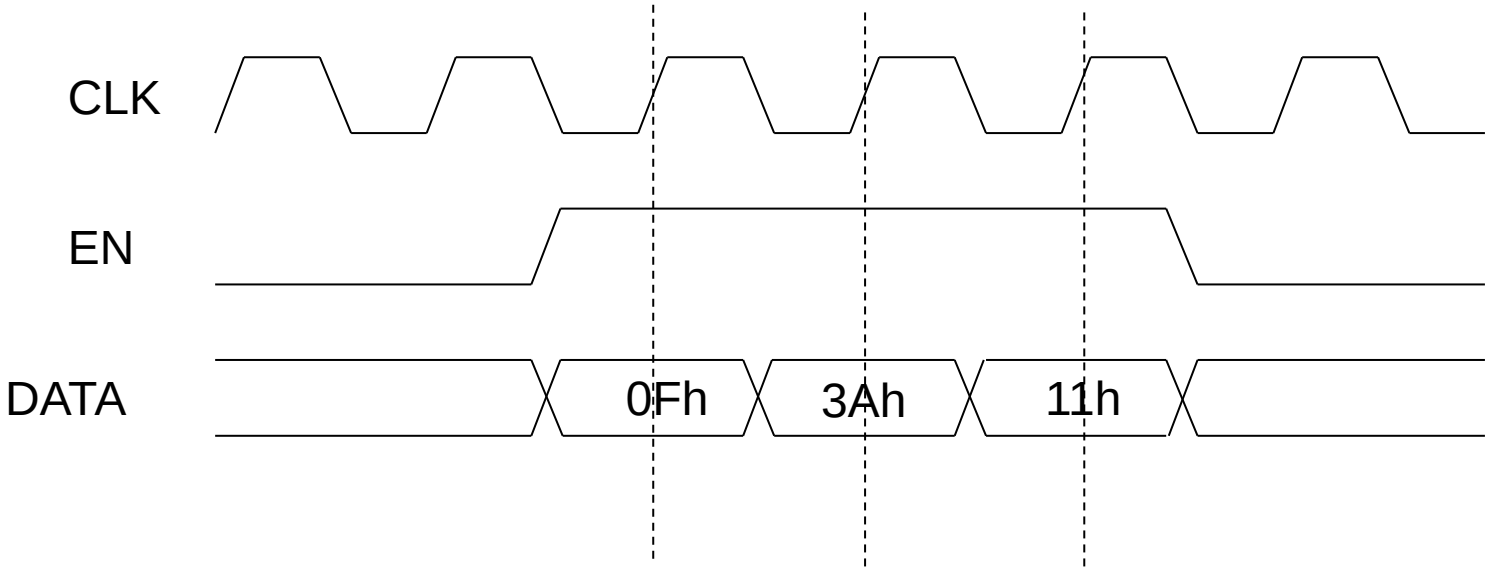
Rozdělení sběrnic dle účelu

- Procesorová
  - DMI (Direct Media Interface, Intel)
  - UMI (Unified Medial Interface, AMD)
  - HyperTransport (AMD)
- Paměťová
  - DDR, DDR2, DDR3
- Systémová
  - PCI, PCI Express
- Periferní
  - SCSI (SAS), USB, FireWire, SATA, ATA (IDE)

Rozdělení sběrnic dle synchronizace

- Synchronní
  - Přenos dat je synchronizován hodinovým signálem
  - Např. PCI, SMBus, paměťové sběrnice
- Asynchronní
  - Přenos dat řízen potvrzovacími signály typu data připravena, data převzata
  - Např. SCSI

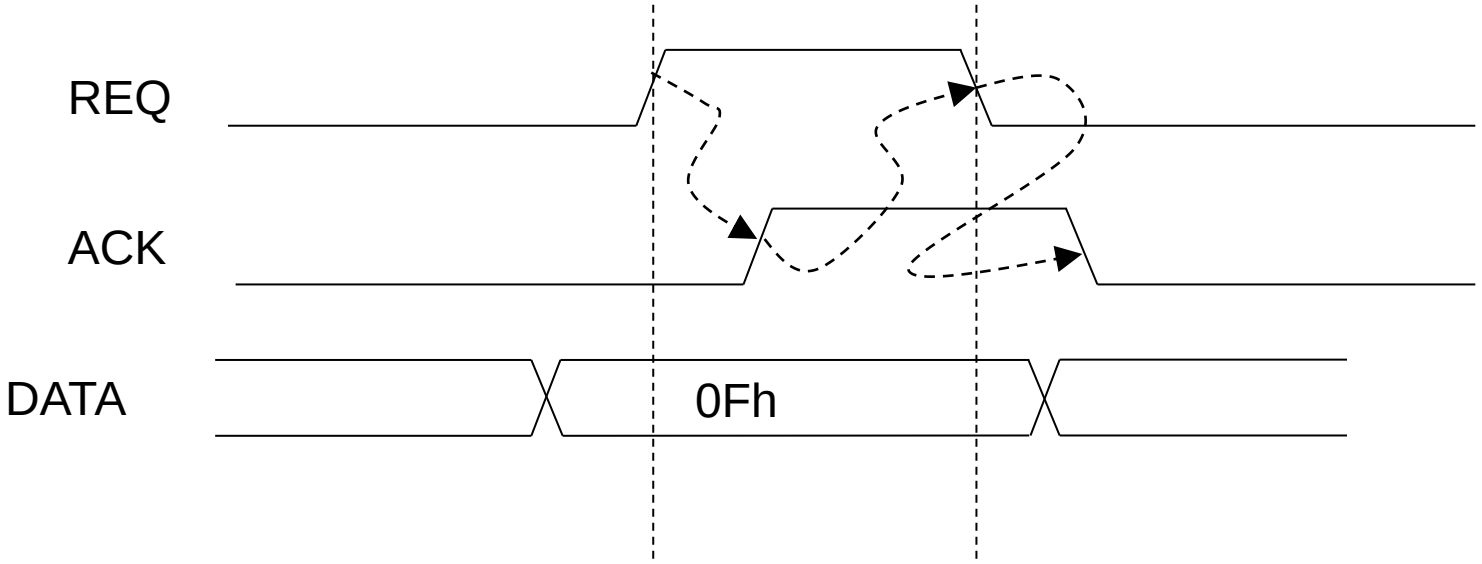
Synchronní přenos dat



Data jsou zapisována do cílového zařízení vzestupnou hranou hodinového signálu. Data jsou tutíž přenášna synchronně s hodinovým signálem, odtud vyplývá synchronnost přenosu. Platnost dat je v tomto konkrétním případě indikována signálem signál EN (Enable). Pokud je signál neaktivní (log. 0) cílové zařízení data ignoruje.

Některé synchronní sběrnice používají k vzorkování dat obě hrany (sestupnou i vzestupnou). Tím se při nezměněné frekvenci hodinového signálu dosahuje dvojnásobné přenosové rychlosti (paměti DDR a disky Ultra DMA).

Asynchronní přenos dat



Zdroj dat vystaví data na datové vodiče a vzestupnou hranou signálu REQ (Request) indikuje cílovému zařízení platnost dat. Vzestupná hrana signálu REQ se může přímo použít pro zápis dat to cílového zařízení. Po zápisu dat indikuje cílové zařízení signálem ACK převzetí dat. Zdroj dat zareaguje deaktivací signálu REQ a zneplatněním dat na datových vodičích. Po deaktivaci signálu REQ deaktivuje cílové zařízení signál ACK.

Tento typ přenosu využívá např. SCSI.

Rozdělení sběrnic dle směru přenosu dat

- Jednosměrné
  - Data přenášena pouze v jednom směru
  - Typický příklad: adresová sběrnice
- Obousměrné
  - Data přenášena oběma směry
  - Lze přenášet v daném časovém okamžiku data pouze v jednom směru

Rozdělení sběrnic dle způsobu přenosu dat

- Paralelní
  - Data se přenášení najednou (paralelně) v dané šíři, např. 8 bitů, 16bitů, 32bitů, …
  - U vysokých přenosových rychlostí hraje roli vliv doba šíření signálu od zdroje k cíli. Různá délka vodičů na motherboardu působí, že jednotlivé bity dorazí k cíli v jiný čas. To limituje maximální přenosovou rychlost. Řešením je sériový přenos dat.
  - Příklad: PCI, paměťové sběrnice, SCSI
- Sériové
  - Data se přenášejí po bitech za sebou (sériově)
  - Mezi komunikujícími obvody je jen minimum vodičů
  - Vhodnými budiči lze dosáhnout vysokých přenosových rychlostí
  - Z více sériových linek lze složit "paralelní", pozor ! Data se ale na jednotlivých linkách přenášejí nezávisle (nejsou vzájemně synchronizovány, jako na paralelní sběrnici)
  - Příklady: USB, FireWire, PCI Express, SATA

Metody připojování periferií

BI-MPP

Přednáška 2

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Adresní prostory PC
- Překlad adres
- V/V operace
- Přímý přístup do paměti

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2

Adresní prostor (opakování)

- Souvislý rozsah adres generovaný procesorem
- Velikost dána počtem adresních bitů, velikost je vždy mocnina dvou
- Nejmenší adresovatelná datová jednotka v adresním prostoru může být:
  - 1 bit (ve speciálních případech)
  - 1 slabika (1 byte), nejčastější (PC)
  - 1 slovo 16-bitů, 32-bitů, 64-bitů atp.
- Do adresního prostoru se mapuje fyzická paměť (tj. paměť v paměťových čipech) nebo registry
- Adresní prostor nemusí být vždy celý vyplněn fyzickou pamětí
- Pokud procesor podporuje více adresních prostorů, neznamená to, že každý prostor bude mít svou vlastní adresovou sběrnici. Adresní prostory obvykle sdílí jednu adresovou sběrnici a jednotlivé adresní prostory jsou odlišeny oddělenými čtecími a zápisovými signály

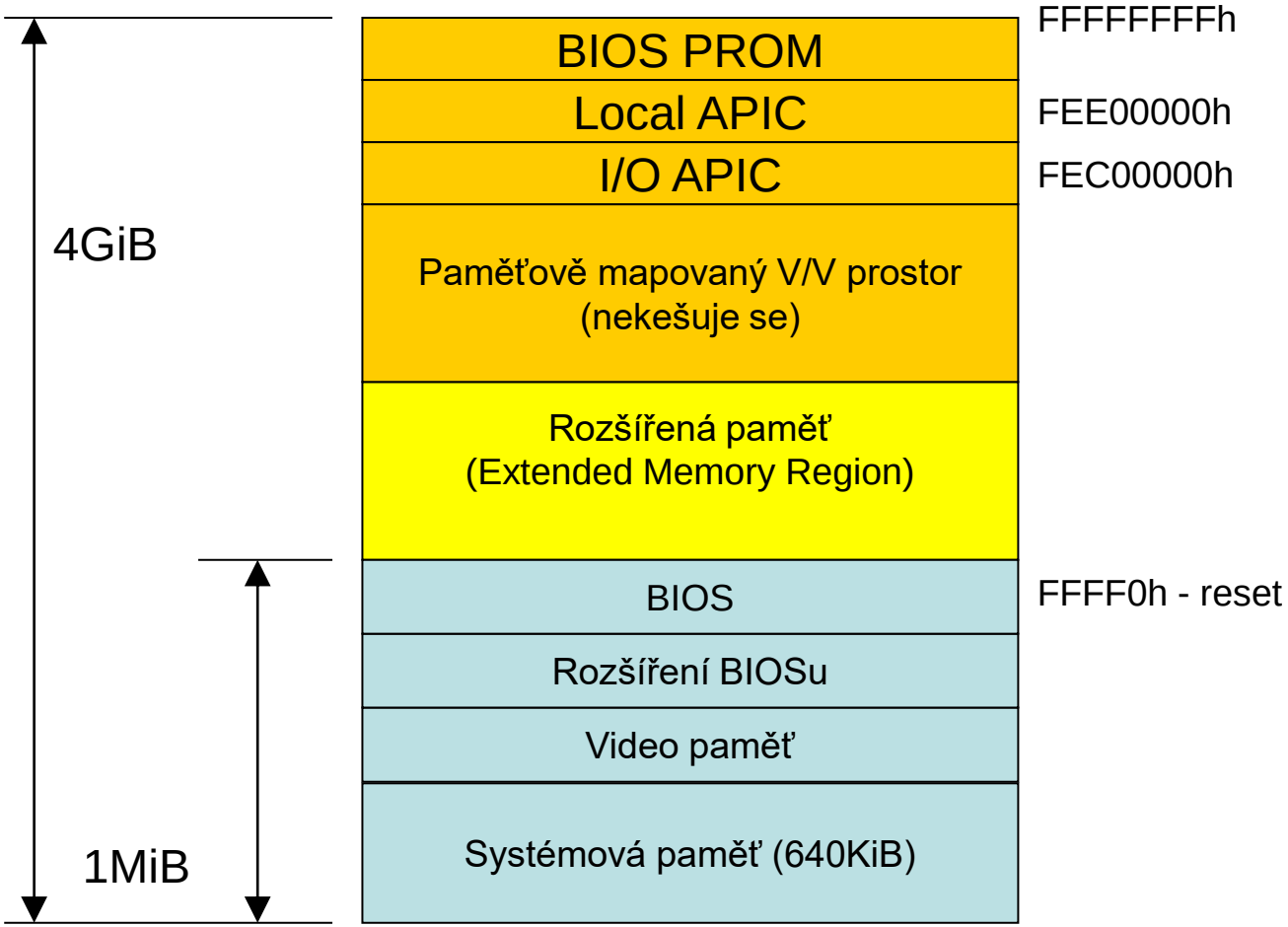
Adresní prostory PC

- Fyzický paměťový prostor
  - Do tohoto prostoru jsou mapovány paměťové moduly
- Virtuální paměťový prostor
  - V tomto prostoru adresují aplikace
  - Každá aplikace má svůj adresní prostor
  - Operace čtení a zápis v tomto prostoru podléhají překladu adres (v pořadí segmentace, stránkování)
  - Do virtuálního adresního prostoru se
    - mapují části fyzické paměti
    - části fyzické paměti odložené na disku (stránkovací soubor, swap partition)
    - Zbytek je neobsazen a jakýkoliv přístup vede k výjimce
- Vstupně/výstupní
  - Velikost 64KiB,mapují se sem registry periferií

Fyzický adresní prostor

- V tomto adresním prostoru je mapována operační paměť
- Na 32 bitové architektuře má tento prostor velikost 4GiB
- Do tohoto adresního prostoru se mapují paměťové bloky periferií (např. videokarty)
- K fyzickému paměťovému prostoru má přístup pouze jádro operačního systému
- Operační systém mapuje části fyzického adresního prostoru do virtuálních prostorů aplikací

Mapa fyzické paměti PC



Virtuální paměťový prostor

- Typicky paměťový adresní prostor přiřazený aplikacím.
- Do tohoto prostoru se mapují části fyzického adresního prostoru metodou
  - Segmentace
  - Stránkování
  - Segmentací s následným stránkováním
- Překlad adres z virtuálního paměťového prostoru do fyzického je hardwarový
  - Překládá se na základě překladových tabulek
  - Tabulky upravuje operační systém

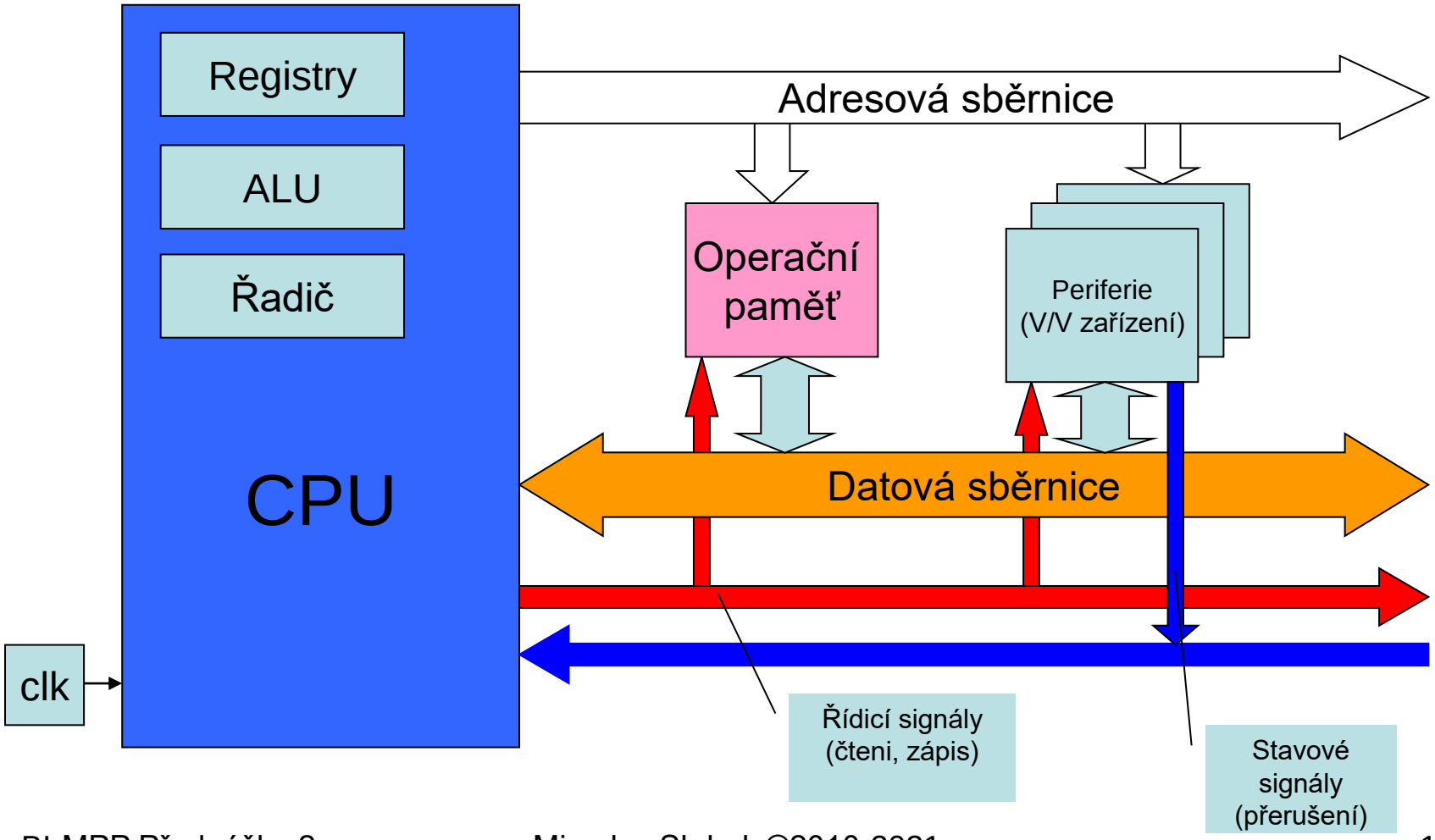
Segmetace

- Segment je část virtuálního (při vypnutém stránkování přímo fyzického) adresního prostoru určená svým počátkem a délkou
- Segment je určen záznamem v tabulce deskriptorů
- Procesor obsahuje segmentové registry, které obsahují ukazatele do tabulky deskriptorů
  - CS registr (Code segment, čtení instrukcí)
  - DS registr (Data segment, proměnné programu)
  - SS registr (Stack segment, zásobník programu)
  - ES, FS, GS registr (další segmentové registry)
- Registry jako IP (Instruction pointer) jsou posunutím (offsetem) v rámci segmentu CS, např. EBX je posuntím DS (MOV DS:[EBX], EAX; MOV [EBX], EAX)
- Při zapnutém stránkování jsou segmenty mapovány do fyzické paměti stránkovacím mechanismem.
- Operační systémy jako Linux obchází segmentaci (nelze totiž vypnout) nastavením segmentů na celý adresní prostor.

Stránkování

- Fyzický adresní prostor je rozdělen na stejně velké stránky (typicky 4KiB)
- Jednotlivé stránky se v různém pořadí mapují do virtuálních prostorů aplikací (souvislý úsek paměti ve virtuálním prostoru nemusí být souvislým úsekem ve fyzické paměti)
- Mapování stránek se provádí přes tabulku stránek, která je umístěna ve fyzické paměti
- Tabulku stránek vytváří a modifikuje operační systém
- Překlad adresy probíhá hardwarově, na jednu operaci např. MOV EAX, [EBX] (čtení dat ve virtuálním prostoru) musí předcházet několik čtecích operací v tabulkách stránek. Pro urychlení překladu se používá asociativní pamět TLB (Translation Lookaside buffer, neplést keší).

Blokové schéma počítače



Zápis a čtení v paměťovém prostoru

Jazyk C

```
long* adr = 0xC0000000;
long v = *adr;
```

Asembler

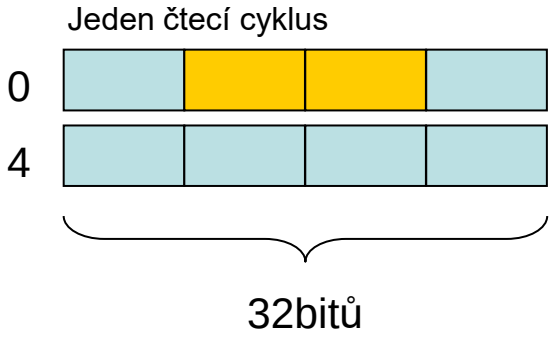
```
MOV EBX, 0xC0000000
MOV EAX, [EBX]
```

Pozor ! Adresujeme ve virtuálním adresovém prostoru

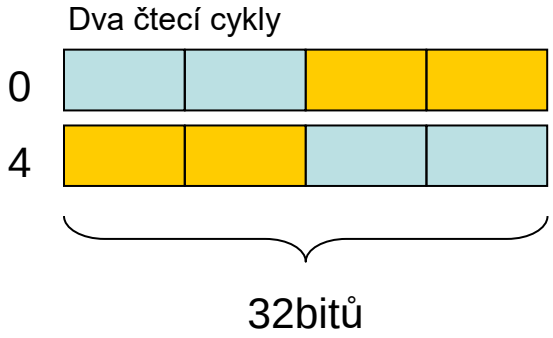
Pokud nejsme v kontextu jádra, ale aplikace, pak to patrně skončí výjimkou, protože pro použitou adresu není mapována fyzická paměť

Nezarovnané V/V a paměťové operace

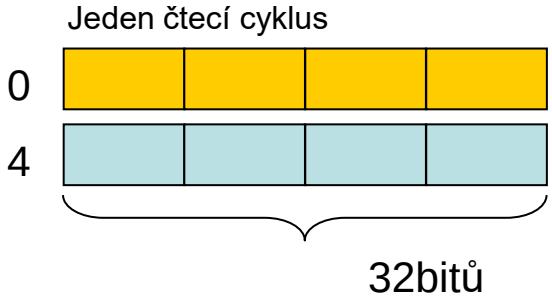
```
mov DX, 1
in AX, DX
```




```
mov DX, 2
in EAX, DX
```




```
mov DX, 0
in EAX, DX
```



Pozor ! Často V/V registry nedovolují nezarovnaný zápis, případně zarovnaný zápis menší jednotky než je 32 bitů.

Příklad: adresy CF8h a CFCh, které se používají pro konfiguraci PCI karet, vyžadují zarovnané 32bitové operace.

Čtení a zápis v IO prostoru

Jazyk C - linux

```
Long v = inb(io_addr)

outb(v, io_addr)
```

Jazyk C – Windows (HAL)

```
READ_PORT_UCHAR(port)

WRITE_UCHAR(port, value)
```

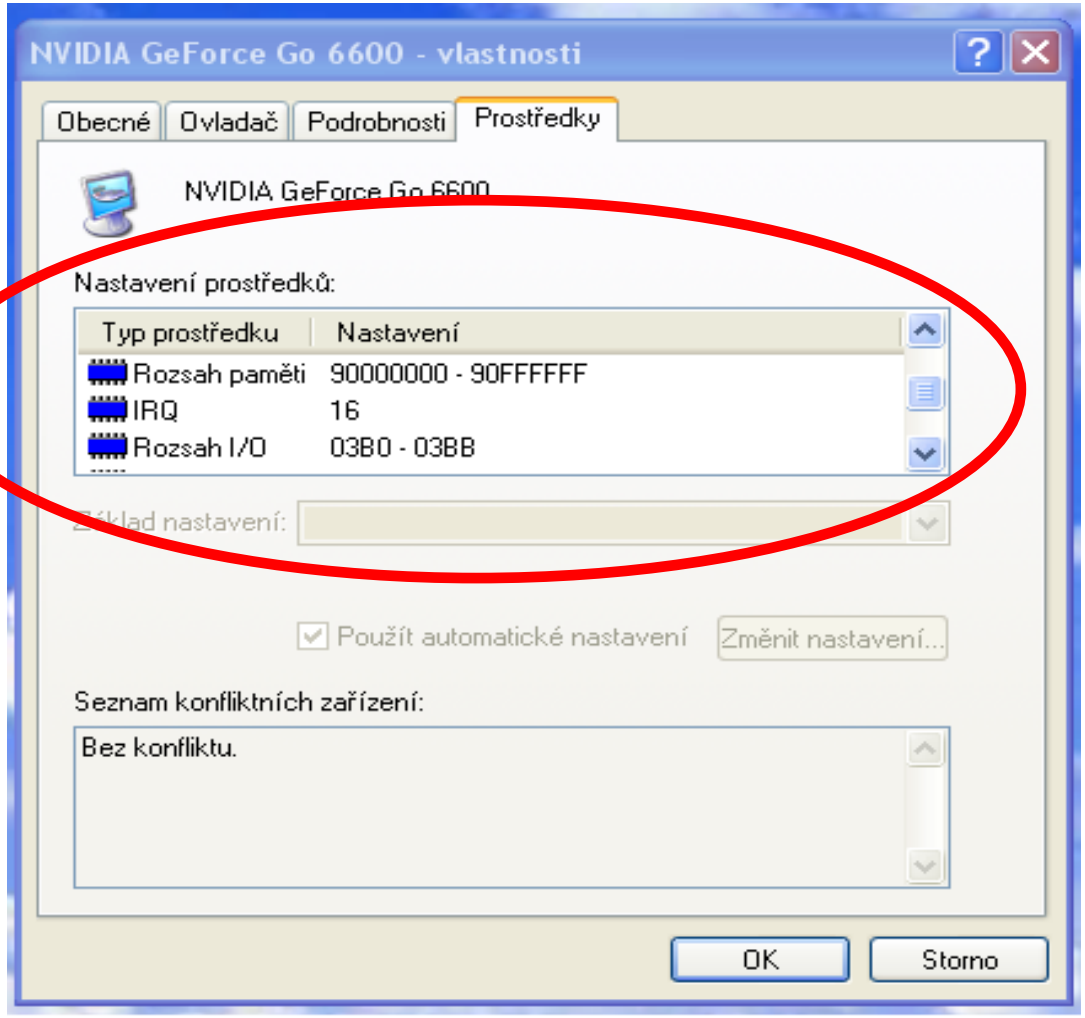
Na úrovni aplikace pokus o tyto operace končí výjimkou.

Na Linuxu je možné povolit funkcí iopl. Ve windows je nutné použít např. giveio

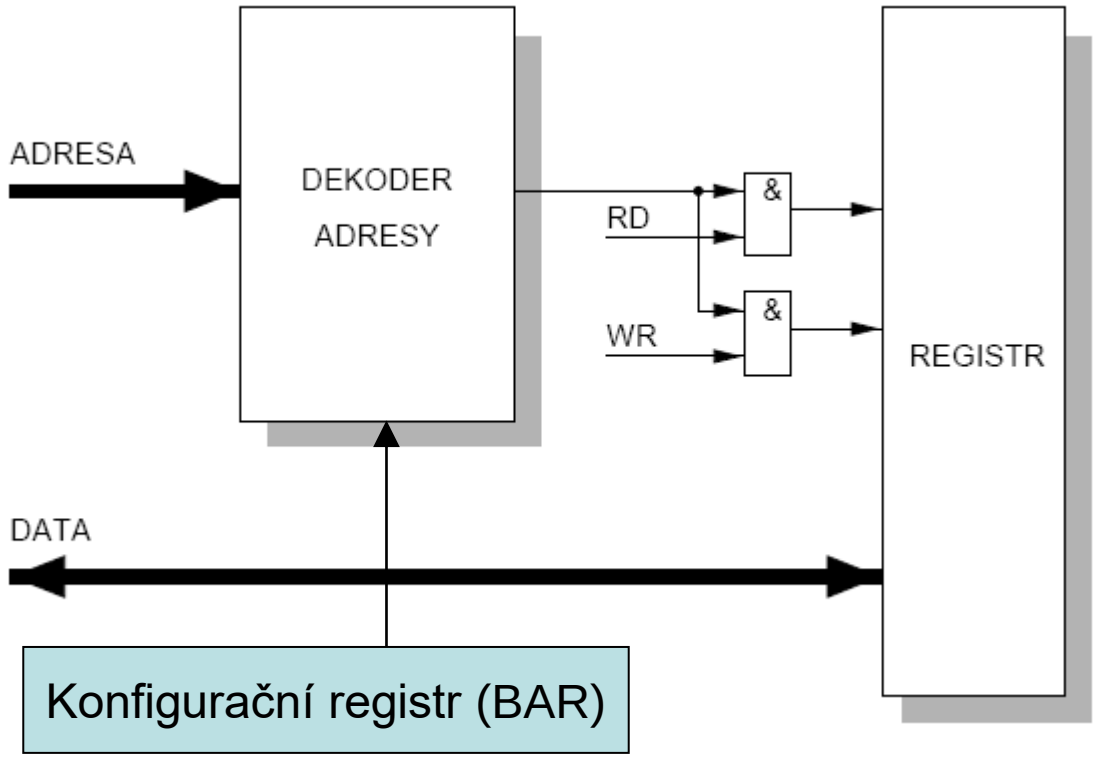
Pozor ! odblokování přístupu do I/O prostoru je v principu bezpečnostní díra !

Při přístupu k portům z aplikace je třeba uvažovat s přepínáním kontextu (např. kdyby někdo chtěl generovat časový průběh)

Prostředky alokované zařízením



Mapování registrů do V/V prostoru

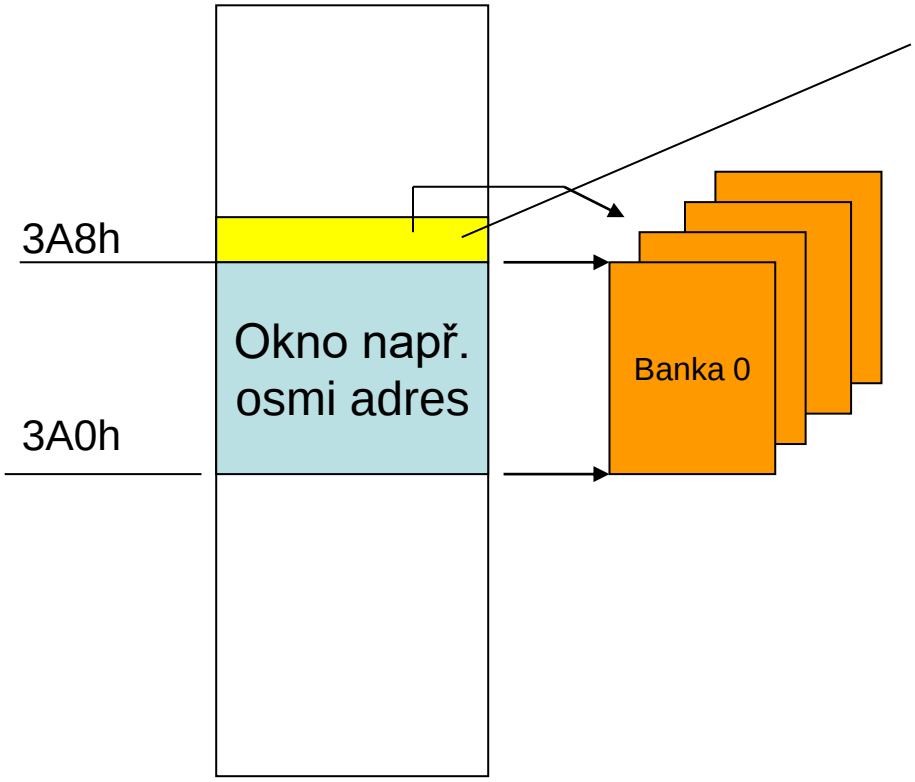


Konfigurační registry PCI karet

Device ID (31..16)		**Vendor ID (15..0)**	
Status (31..16)		Command (15..0)	
Class Code (31..8)		Revison ID (7..0)	
BIST (31..24)	Header Type (23..16)	Latency Timer (15..8)	Cache Line Size (7..0)
Base Address Register 0			
Base Address Register 1			
Base Address Register 2			
Base Address Register 3			
Base Address Register 4			
Base Address Register 5			
Cardbus CIS Pointer			
Subsystem ID (31..16)		Subsystem Vendor (15..0)	
Expansion ROM Base Address			
Reserved			
Reserved			
Max_Lat (31..24)	Min_Gnt (23..16)	Interrupt pin (15..8)	Interrupt line (7..0)

Mapování bank registrů

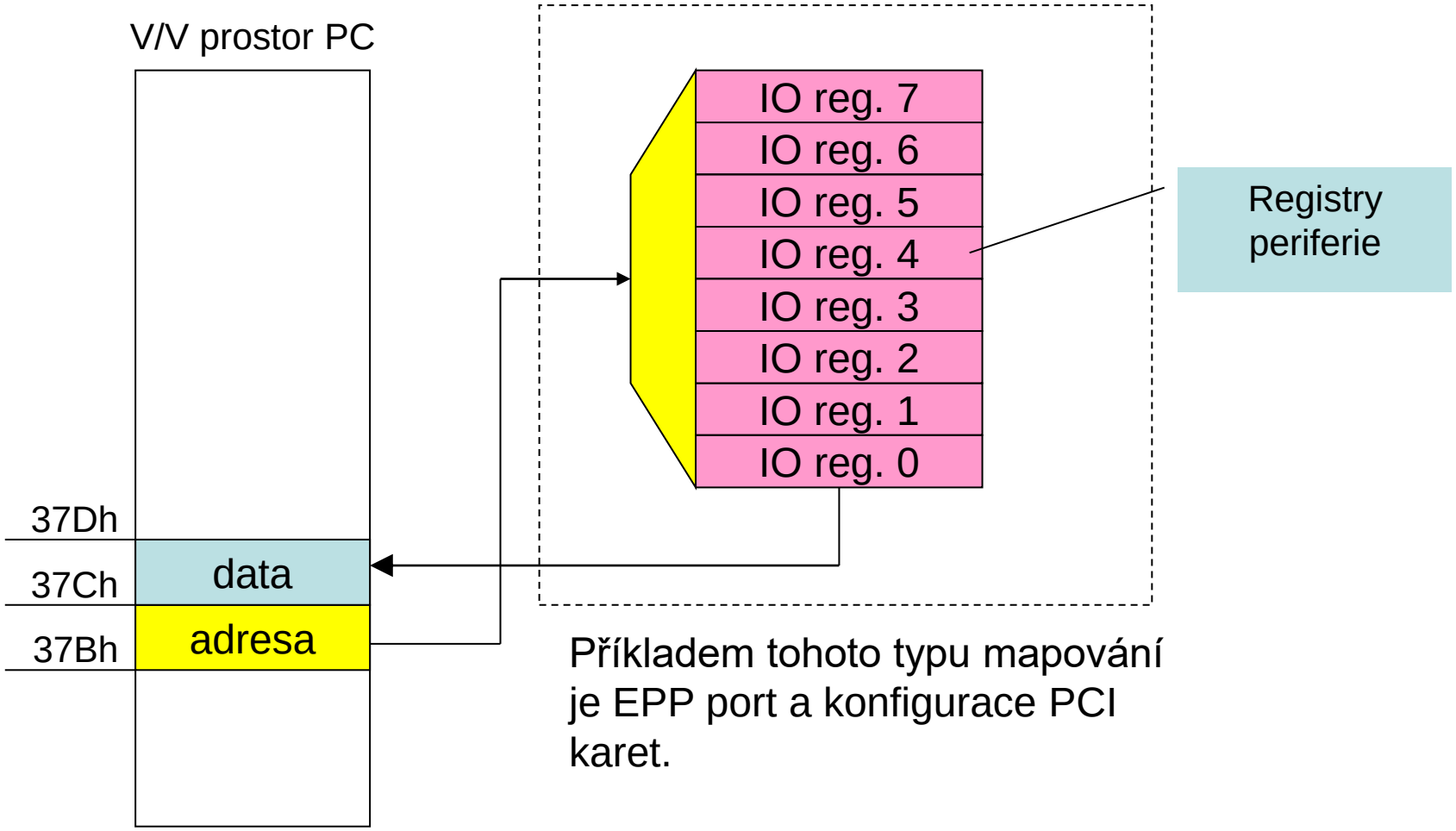
Pokud má periferie větší množství registrů, pak použije metodu mapování bank, aby se ušetřily adresy ve V/V adresním prostoru.



Výběr banky
(v tomto případě stačí zapsat 2 bity do registru na adrese 3A8h pro výběr konkrétní banky registrů)

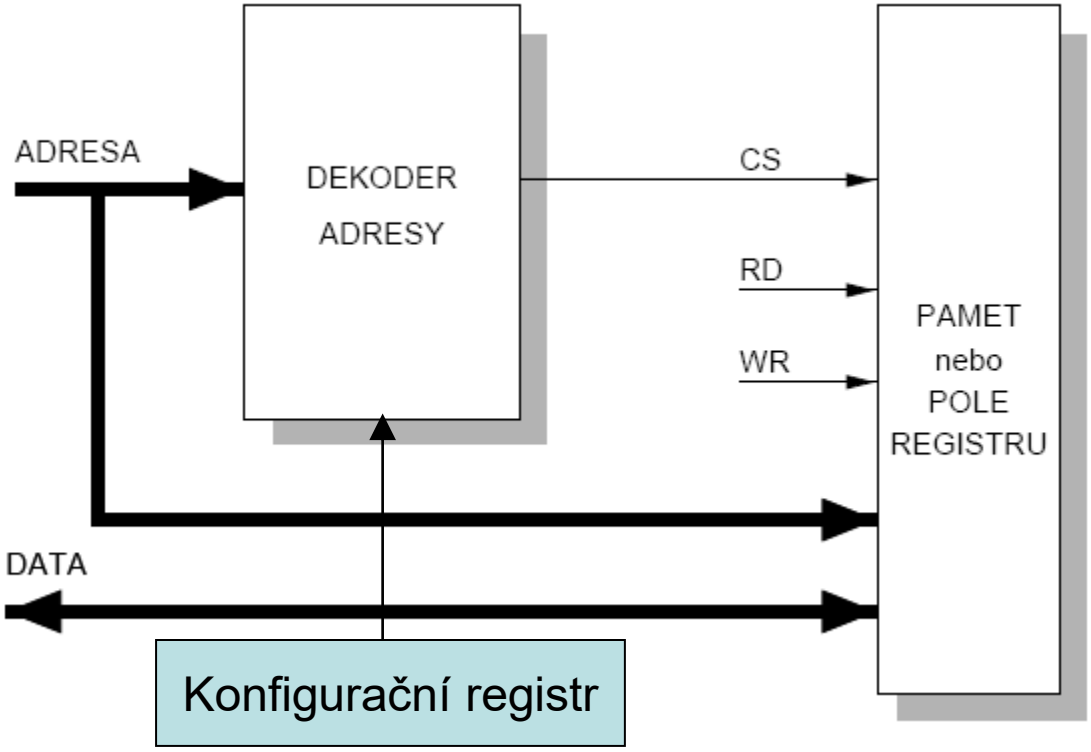
U IrDA řadičů na motherboardu je registr pro výběr banky součástí okna. Pak musí být tento registr mapován ve všech bankách a na stejnou adresu.

Adresační přístup k registrům

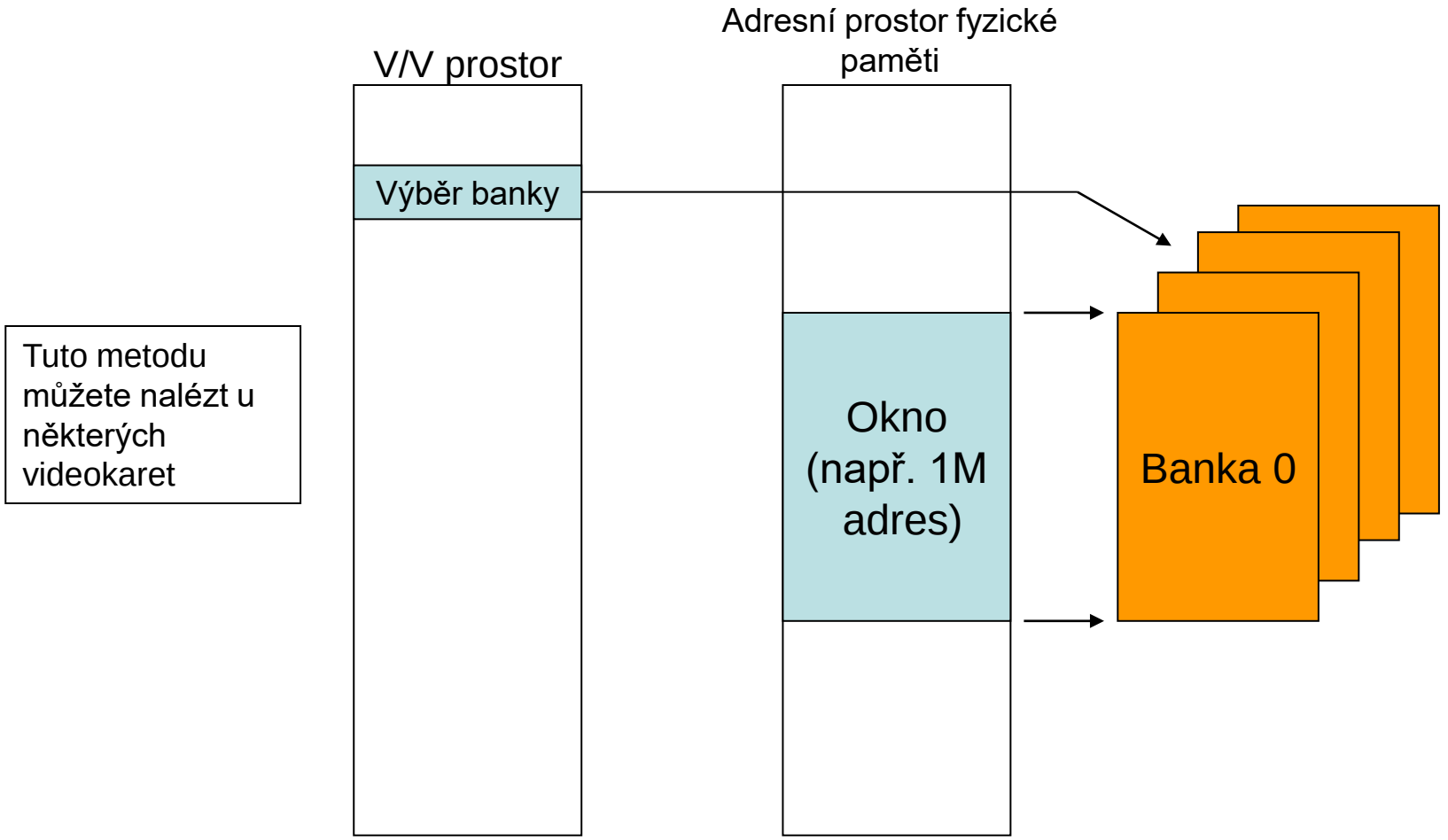


Příkladem tohoto typu mapování je EPP port a konfigurace PCI karet.

Mapování registrů nebo paměti do paměťového adresového prostoru



Mapování paměťových bank



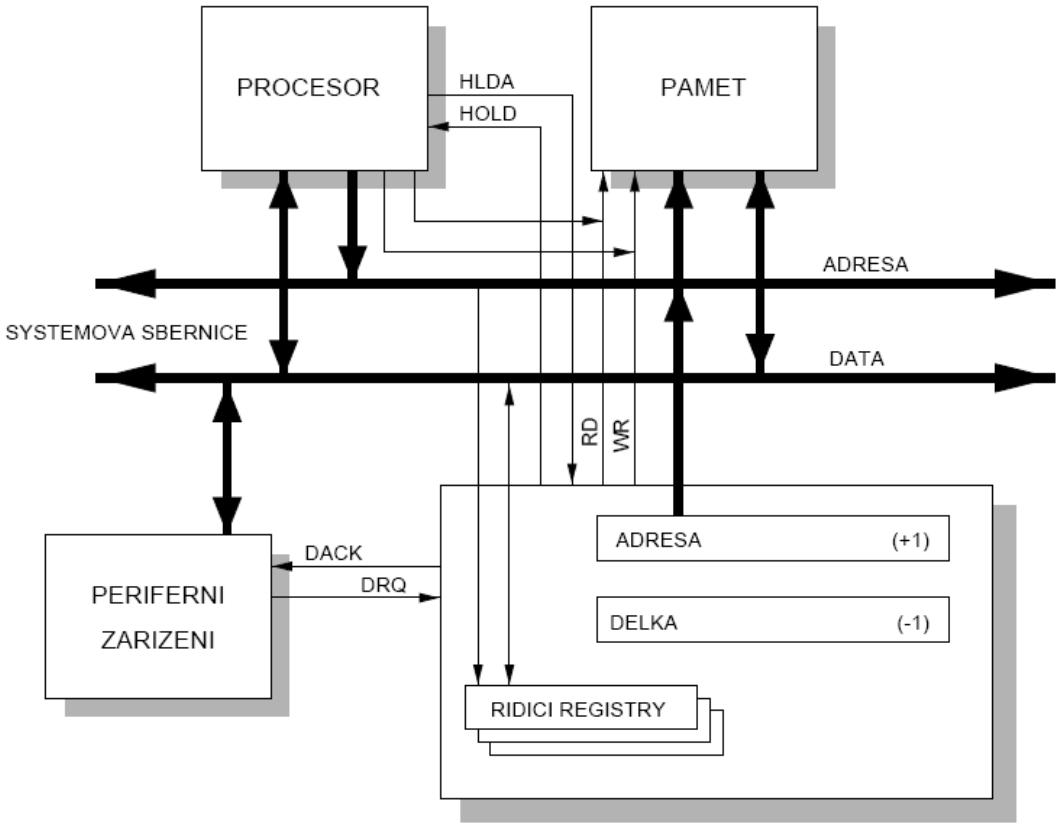
PnP přístupy při mapování registrů a paměťových bloků

- Rozmanitost V/V zařízení nedovoluje předem určit pevné adresy registrů V/V zařízení (velká pravděpodobnost kolize)
- Ruční konfigurace je uživatelsky nepohodlná
- Dnešním řešením je programová konfigurace zařízení (princip PnP)

V/V adresy v PC

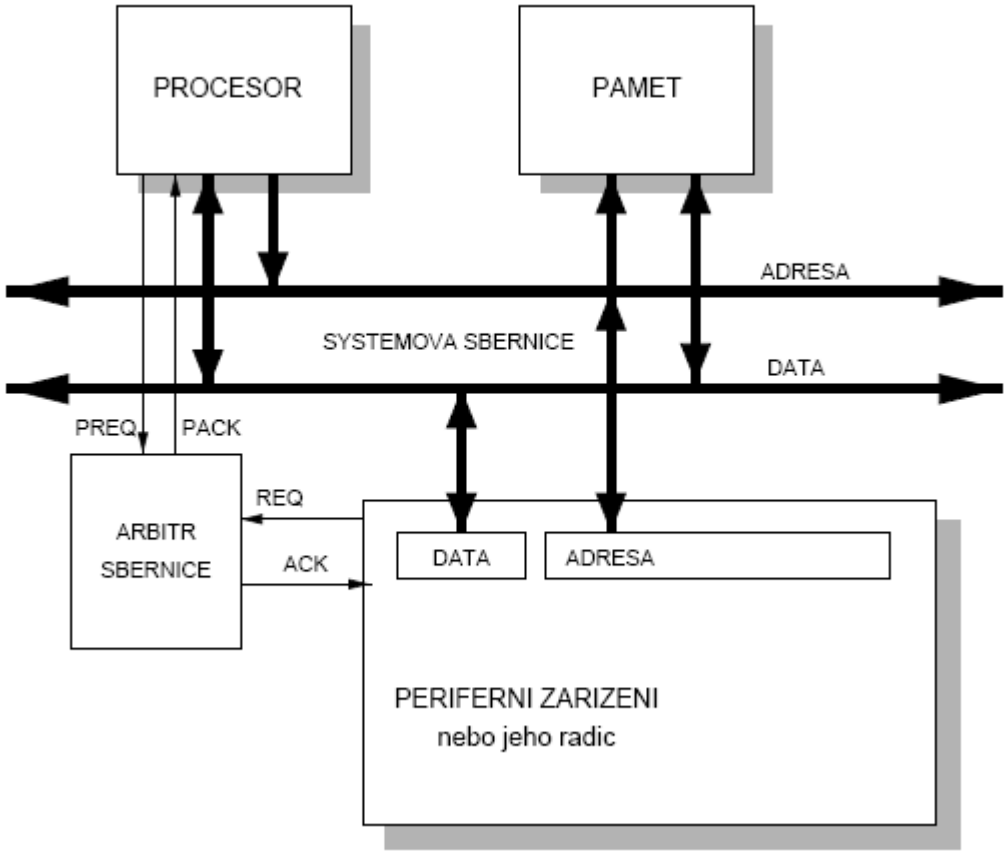
- Pevné – dány historicky, udržovány z důvodu zpětné kompatibility. Např. čítač, původní řadič přerušení, DMA řadiče apod.
- Pevné – zajišťující geografické adresování periferií a přístup do konfiguračních prostorů (typicky PCI karet)
- Konfigurovatelné – bloky adres, které jde umístit téměř libovolně v adresových prostorech. Za technickou realizaci mapování je zodpovědná periferie, za konfiguraci pak setup nebo operační systém.

DMA přenos s externím řadičem



Dnes je již udržováno jen z důvodu kompatibility.

DMA typu Bus-mastering



Metody připojování periferií

BI-MPP

Přednáška 3

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- PCI
- PCI Express
- Přerušení

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- PCI Express™ Base Specification Revision 1.0a. PCI Sig. April 15, 2003

PCI Local Bus

- synchronní systémová sběrnice
- procesorově nezávislá
- hodinový kmitočet 33MHz nebo 66MHz
- šířka dat 32bitů nebo 64bitů, zabezpečeno paritou
- sběrnice typu multi-master
- nízká cena

PCI Local Bus na PC

- Systémová sběrnice: PCI 32 bitů, 33MHz

  určeno širokou škálu V/V karet,
  postupně nahradí sběrnici ISA

- AGP (Accelerated Graphics Port):
  modifikovaná PCI 32 bitů, 66MHz

  určeno pro grafické karty

Signály PCI sběrnice

Adresy a data

AD[0::31]	multiplexovaná datová a adresová sběrnice
C/BE[0::3]#	příkaz/platnost bytu
PAR	parita přes AD a C/BE# signály

Řídící signály

FRAME#	rámec transakce
IRDY#	iniciátor připraven
TRDY#	cíl připraven
STOP#	požadavek cíle na zastavení transakce
LOCK#	indikuje atomické sběrnicové operace
IDSEL	výběr karty při konfiguračních cyklech
DEVSEL#	indikuje, že cíl dekódoval adresu

Signály PCI sběrnice

Systémové signály

CLK	hodiny sběrnice (33MHz nebo 66MHz)
RST#	reset sběrnice

Arbitrace

REQ#	žádost o přidělení sběrnice
GNT#	indikuje, že sběrnice byla přidělena

Chybové signály

PERR#	chyba parity
SERR#	chyba v systému

Přerušení

INTA#, INTB#, INTC# INTD#	urovňová přerušení

Signály PCI sběrnice

Signály pro podporu keší

SBO# indikuje adresa paměťové operace odpovídá modifikované položce v některé keši. Tento signál dovolí keši aktualizovat data v paměti.

SDONE# indikuje, že všechny karty dokočily snooping

Přídavné signály

PRSNT[1:2] indikuje vložení PCI karty do konektory

TCK, TDI, TDO, TMS, TRST# - JTAG/Boundary Scan

Signály PCI sběrnice

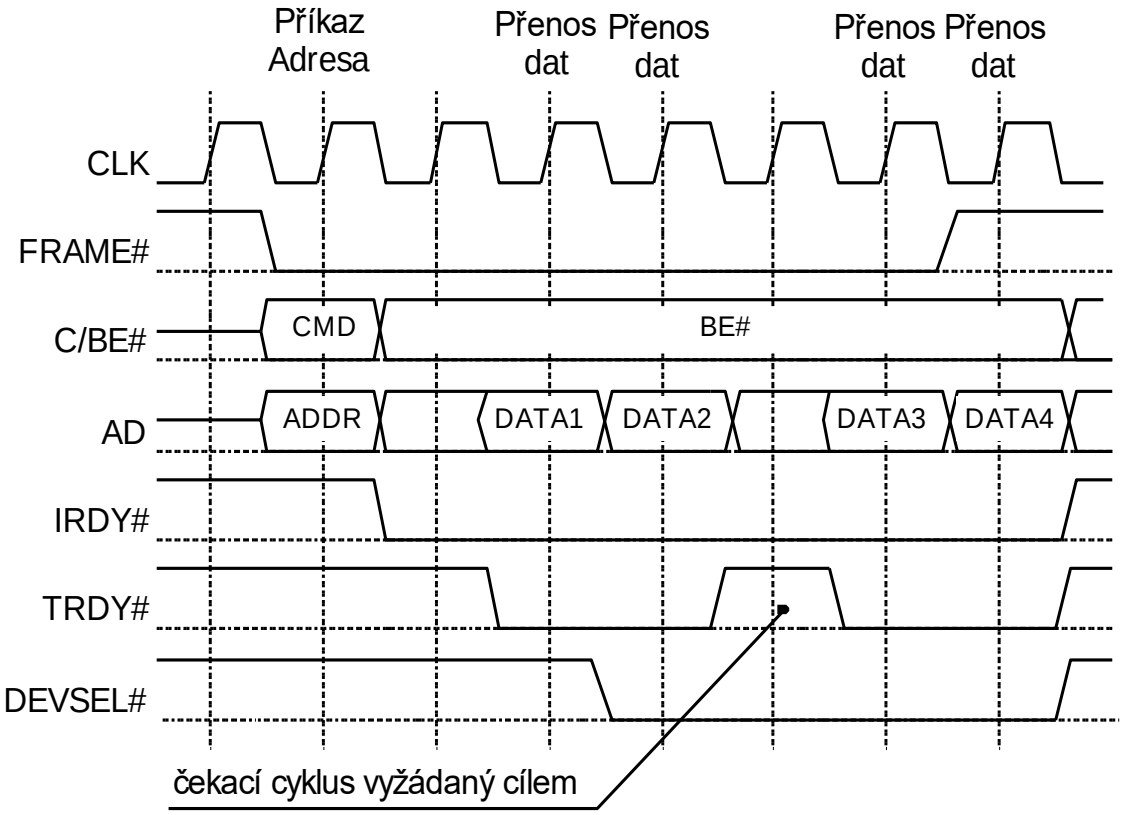
Rozšíření PCI na 64 bitů

AD[63::32]	rozšíření adresové a datové sběrnice
CBE[7::4]#	rozšíření příkazu a signálů a platnosti dat
REQ64#	požadavek na 64 bitový přenos dat
ACK64#	potvrzení 64 bitového přenosu dat
PAR64	parita pro bity [63::32]

Příkazy (C/BE[3:0]#)

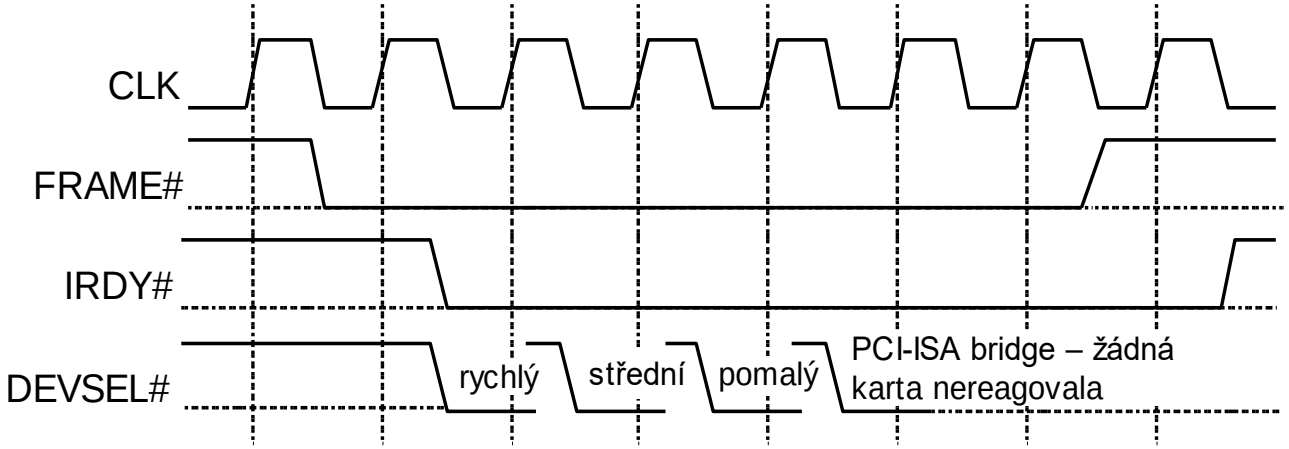
0000	potvrzení přerušení (Interrupt Acknowledge)
0001	speciální cyklus (Special Cycle)
0010	čtení z V/V prostoru (IO Read)
0011	zápis do V/V prostoru (IO Write)
0110	čtení z paměti (Memory Read)
0111	zápis do paměti (Memory Write)
1010	čtení z konfiguračního prostoru (Configuration Read)
1011	zápis do konfiguračního prostoru (Configuration Write)
1100	vícenásobné čtení z paměti (Memory Read Multiple)
1101	přenos 64 bitové adresy (Dual Address Cycle)
1110	čtení z paměti s ohledem na keše
1111	zápis do paměti s ohledem na keše (Memory Write and Invalidate)

Časování PCI sběrnice



Klasifikace adresových dekodérů

Adresové dekodéry se rozlišují podle zpoždění signálu DEVSEL# na rychlý, střední a pomalý



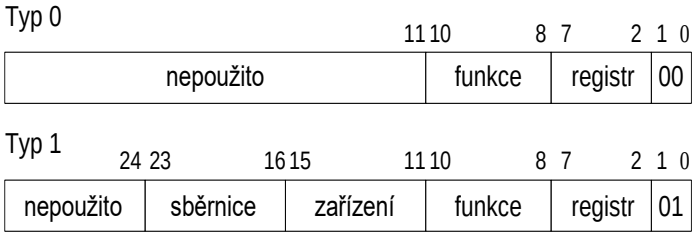
Speciální cykly

- používají se pro přenos speciálních informací mezi zařízeními jedné sběrnice
- adresa pozbývá svůj význam (broadcast)
- datové přenosy reprezentují přenosy jednotlivých zpráv
- 32-bitová data jsou rozdělena na kód zprávy AD[0::15] a datovou část zprávy AD[15::31]

Konfigurační cykly

- umožňují přístup ke konfiguračním registrům karet
- adresace PCI zařízení je v konfiguračních cyklech vázána na sloty (konektory)
- PCI zažízení může obsahovat více než jednu sadu konfiguračních registrů (multifunkční zařízení).

Adresová část konfiguračního cyklu



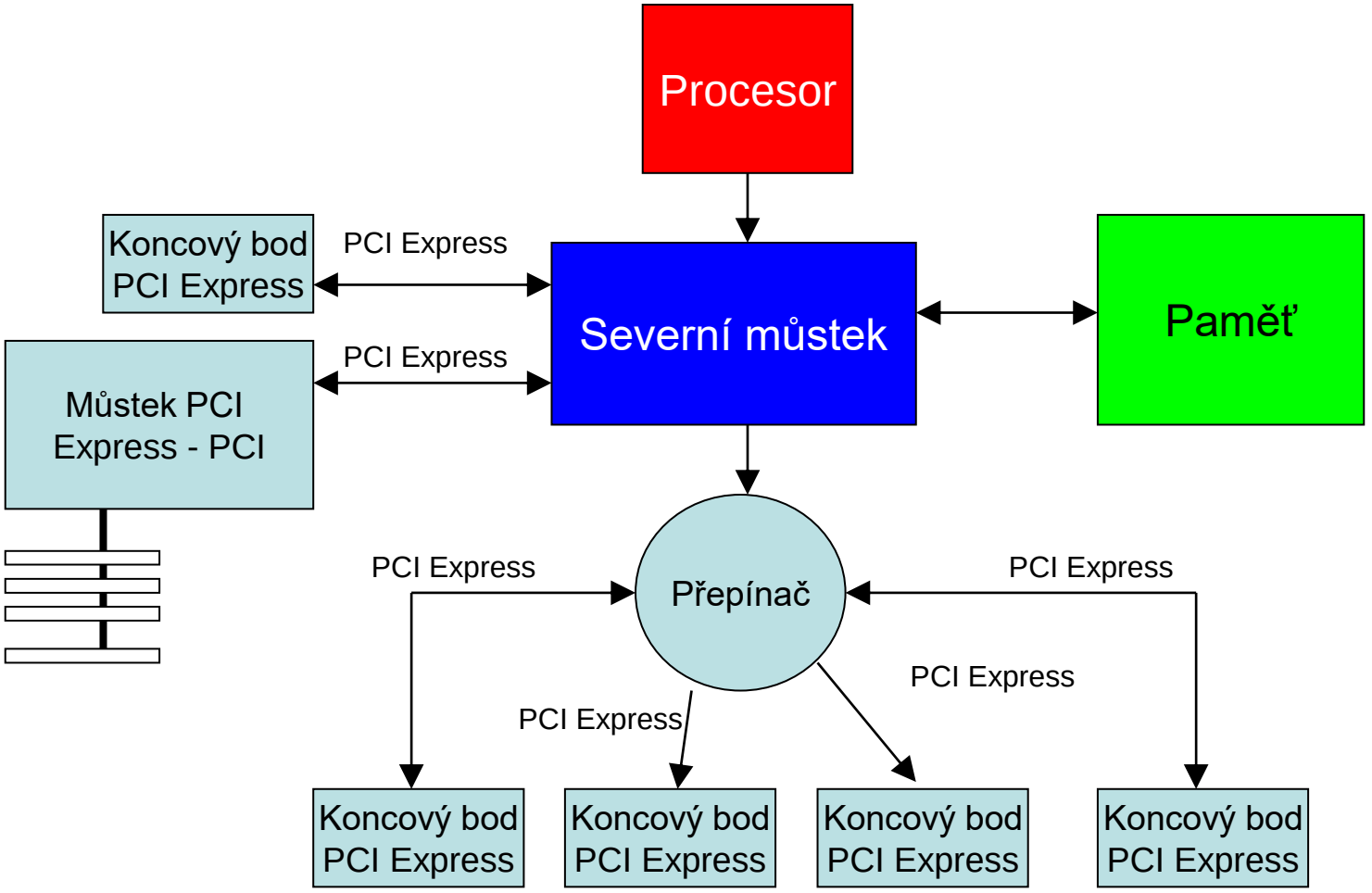
Typ 0 se používá k adresaci zařízení v rámci jedné sběrnice
Typ 1 se používá pro adresaci zařízení za PCI-PCI můstkem

Adresa pro konfigurační adresní prostor se zapisuje do registru na adrese CF8h ve V/V prostoru, těsně před vygenerováním konfiguračního cyklu zápisem/čtením z adresy CFCh. Detaily viz. cvičení.

PCI Express

- Sériová verze PCI, nahrazuje PCI
- Není sběrnicí v pravém slova smyslu, jde o dvoubodový spoj
- Přenosové rychlosti (PCI Express 2.0)
  - 1× - 500 MB/s (obousměrně 1 GB/s)
  - 4× - 2 GB/s (obousměrně 4 GB/s)
  - 8× - 4 GB/s (obousměrně 8 GB/s)
  - 16× - 8 GB/s (obousměrně 16 GB/s)

Architektura PC s PCI Express



Transakce na sběrnici PCI Express

- Paměťové
  - Čtení/zápis do paměti (fyzického paměťového prostoru)
- Vstupně/výstupní
  - Čtení/zápis v 64KiB V/V prostoru
- Konfigurační
  - Čtení/zápis v konfiguračním prostoru
- Výměna zpráv

Transakce v paměťovém prostoru

- Operace čtení zápis
  - ReadRequest/Completion – žádost o čtení a dokončení
  - WriteRequest – zápis
- Transakce mají dva formáty
  - Adresa 32 bitů
  - Adresa 64 bitů

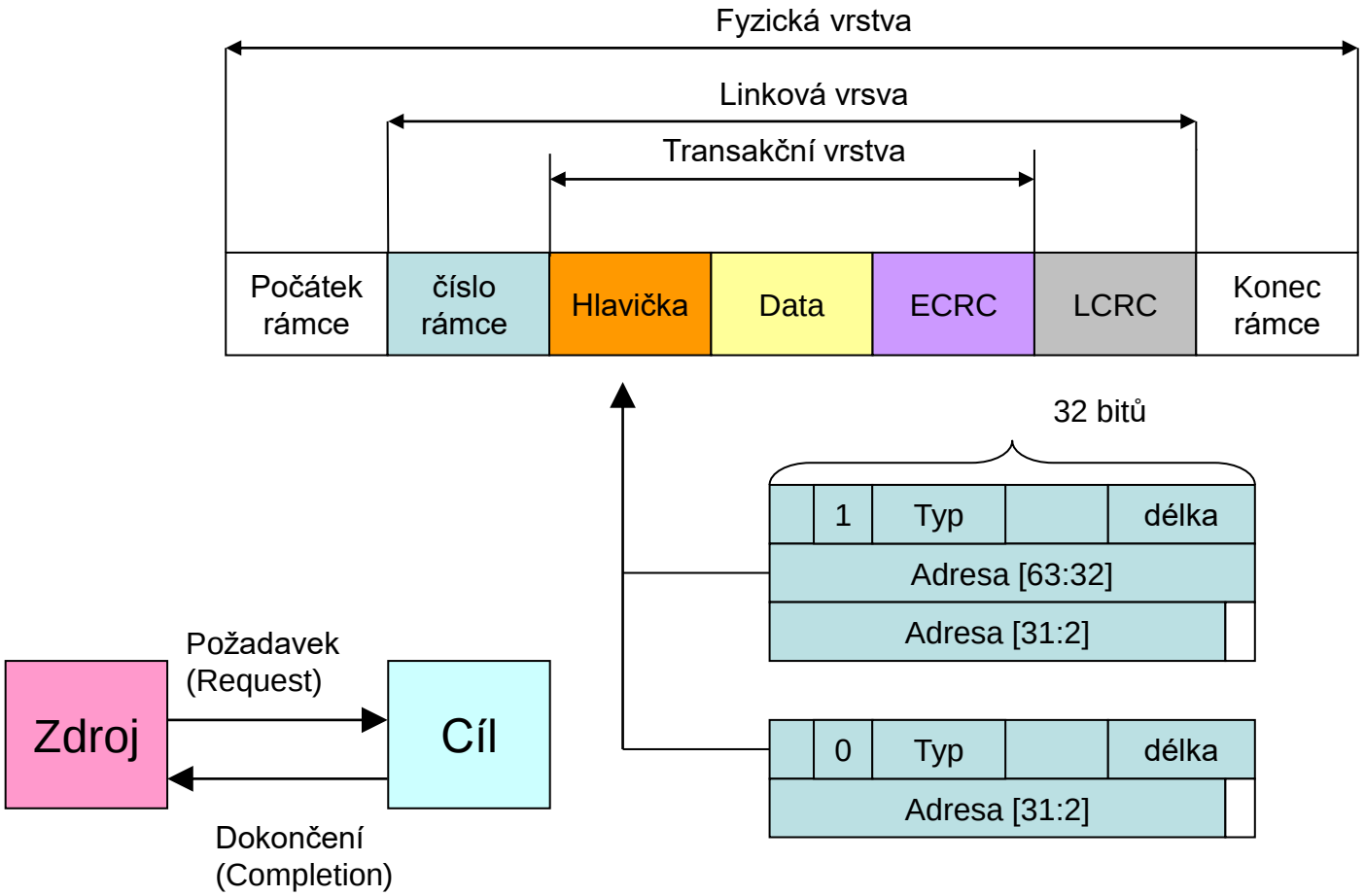
Transakce v V/V prostoru

- Operace čtení/zápis
  - ReadRequest/Completion – žádost o čtení/dokončení
  - WriteRequest/Completion – žádost o zápis/dokončení
- Tento typ transakce používá krátký 32bitový adresační formát

Transakce v konfiguračním prostoru

- Operace čtení/zápis
  - ReadRequest/Completion – žádost o čtení/dokončení
  - WriteRequest/Completion – žádost o zápis/dokončení

Formát paketu



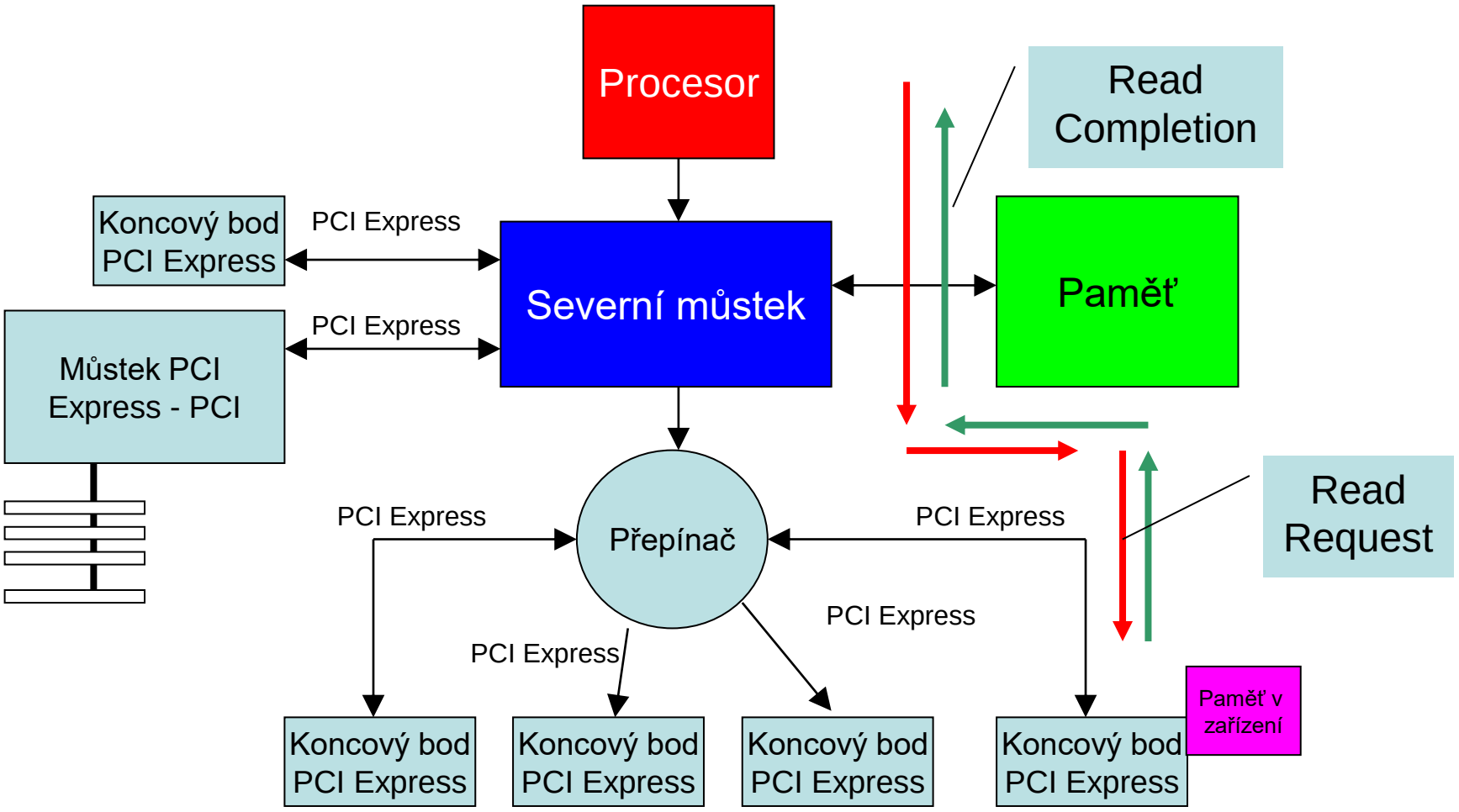
Typy paketů

	Format	Typ	Popis
MRd	00/01	00000	Memory Read Request
MWr	10/11	00000	Memory Write Request
IORd	00	00010	IO Read Request
IOWr	10	00010	IO Write Request
CfgRd0	00	00100	Configuration Read, typ 0
CfgWr0	10	00100	Configuration Write, typ 0
Cpl	00	01010	Completion (bez dat)
CplD	10	01010	Completion (s daty)

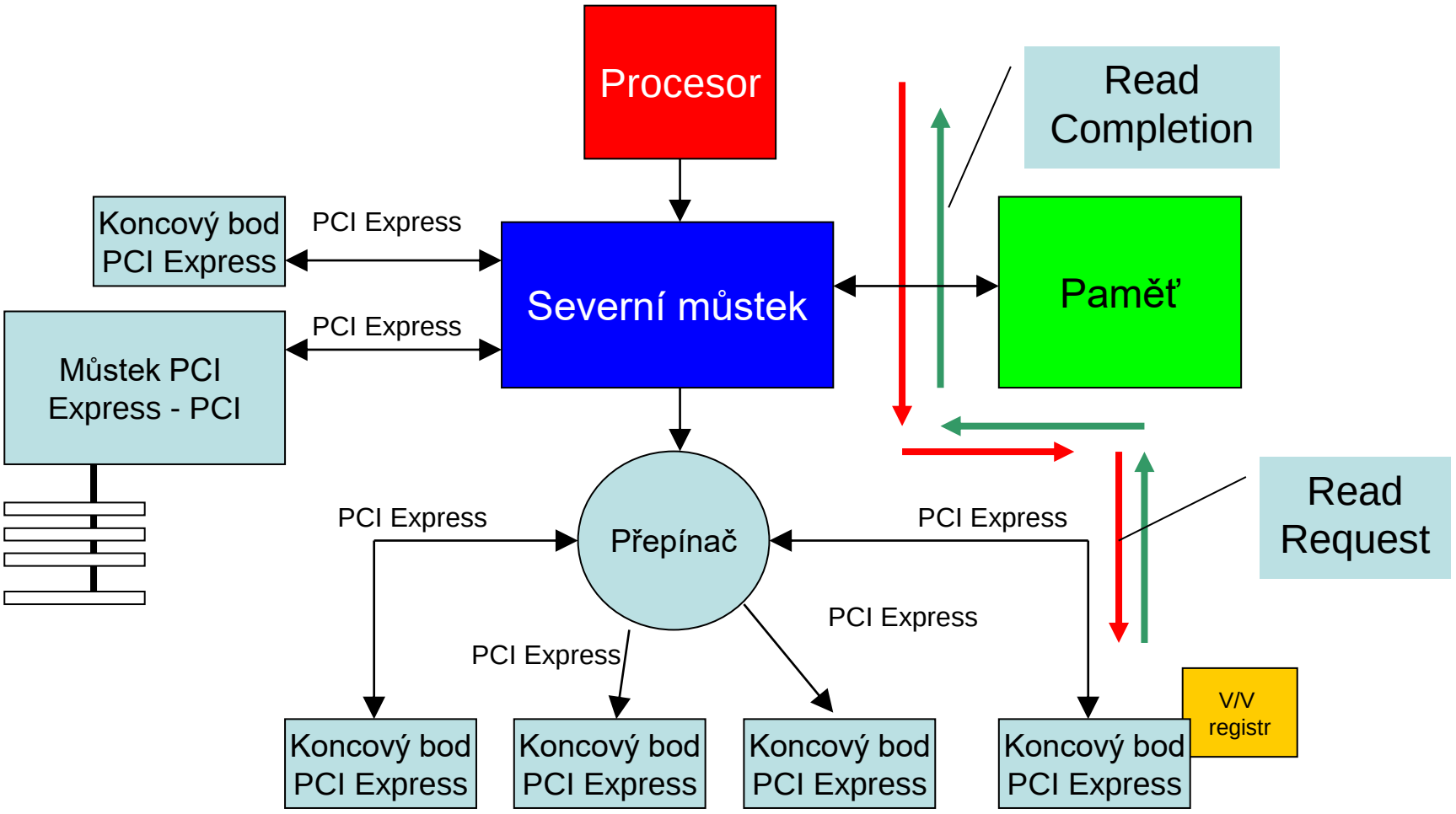
Pozn.: tabulka neobsahuje úplný výčet typů paketů, obsahuje pouze základní typy. Úplnou tabulku naleznete ve specifikaci sběrnice.

Typy paketů odpovídají typům transakcí na sběrnici. Navíc je zde sada dokončovacích paketů, které nesou data nebo stavovou informaci.

Čtení/zápis do paměti

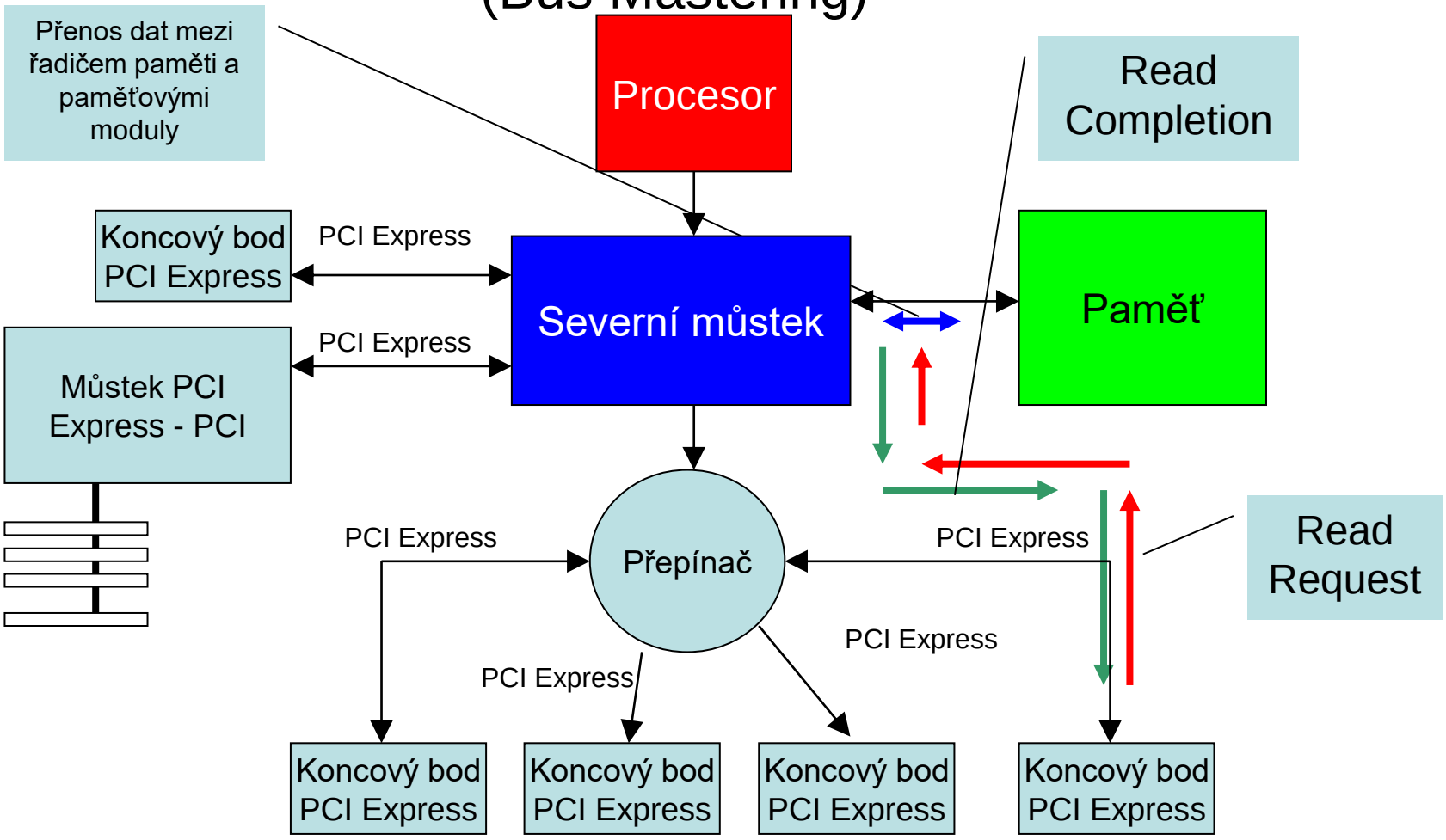


Čtení z V/V registru



Přímý přístup do paměti

(Bus Mastering)



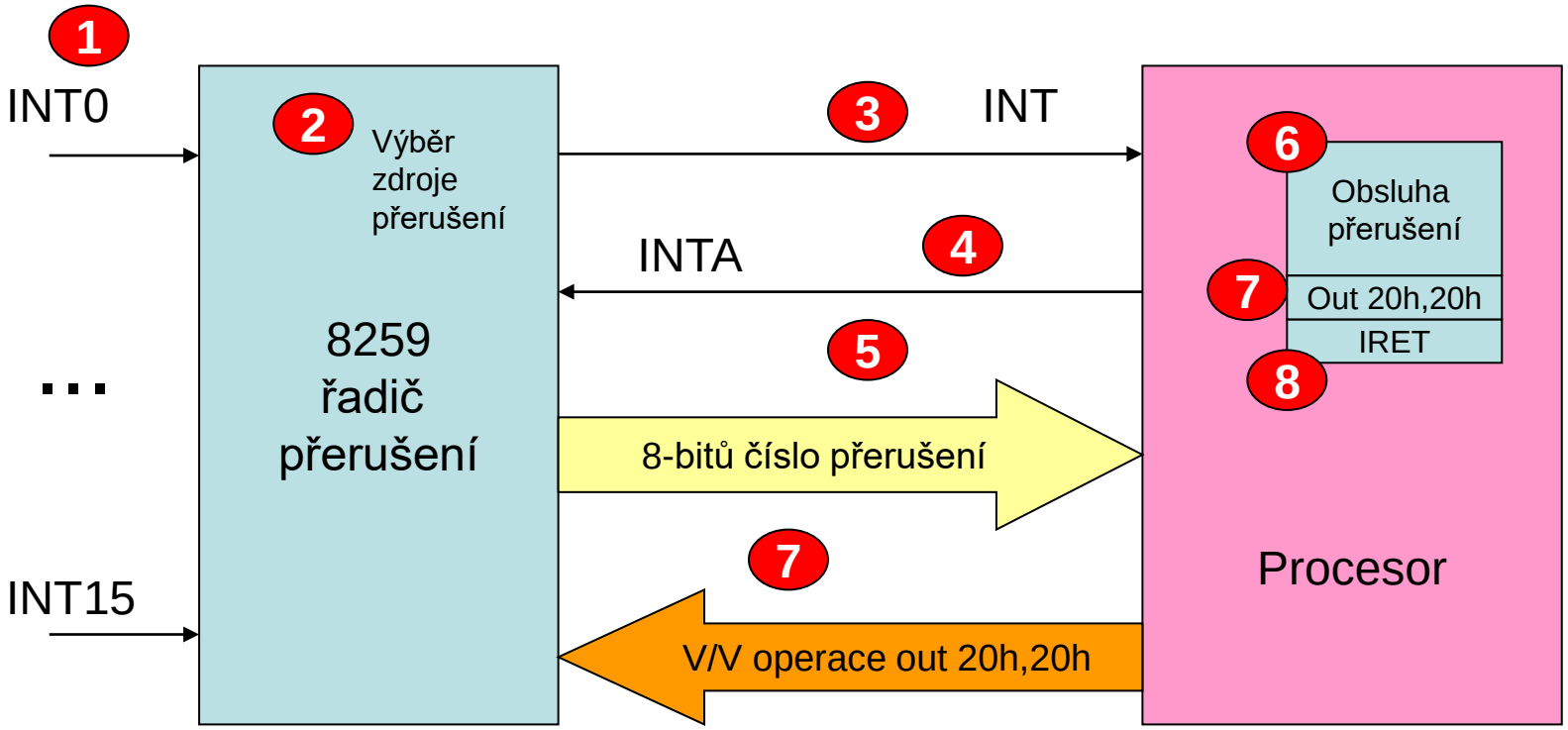
Přerušení

- Mechanismus přerušení zajišťuje vyvolání podprogramu (tzv. obsluhy přerušení) na základě vnějšího signálu nebo vnitřní chyby.
- V počítačích PC lze rozlišit až 256 zdrojů přerušení, prakticky se používá pouze 15. Ostatní lze vyvolat instrukcí **int** n (např. int 21) a používá se jako vstupní body do jádra operačního systému.
- Rozlišení zdrojů a vyhodnocení priorit přerušení zajišťuje řadič přerušení
- IRET, IRETD – instrukce pro návrat z přerušení, kromě návratu obnovuje příznakový registr. V protected modu může způsobit i přepnutí na jiný task.

Řadič přerušení 8259

- Má 15 vstupů přerušení (interrupt 0 – 15)
- Rozlišuje priority přerušení
- Má výstup INT, který signalizuje procesoru žádost o přerušení
- Má konfigurační registry, které jsou mapovány do v/v adresního prostoru
- Ve speciálním sběrnicovém cyklu si procesor žádá od řadiče číslo přerušení, pro které má vyvolat obslužný podprogram
- Pro zpracování další žádosti o přerušení (pokud v daný okamžik existuje) je nutná intervence ze strany obslužného podprogramu, a to zápisem do konfiguračního registru řadiče typicky outb(20h, 20h) se řadiči signalizuje ukončení přerušení.

Interakce procesor ⇔ řadič přerušení



Vektory přerušení a IDT

- V reálném módu procesoru (dnes se používá jen pro start počítače) jsou adresy vstupních bodů přerušení uloženy v tabulce přerušení v 1. KiB operační paměti (256 položek x 4B).
- V protected módu (dnes běžně užívaný) jsou vstupní body obsluh přerušení uloženy jako dekriptory v tabulce IDT (interrupt descriptor table)
- Výběr položky v tabulce vektorů přerušení (případně IDT) provádí procesor na základě čísla přerušení, které dostane ve speciálním cyklu od řadiče přerušení

Přerušení dle maskovatenosti

- Maskovatelné přerušení
  - Přerušení, které lze zakázat globální maskou přerušení (bit IF ve Flags registu). Ovládá se instrukcemi sti (povolení), cli (zákázání přerušení)
- Nemaskovatelné přerušení (NMI)
  - Používá se pro ošetření kritických událostí, například chyby v hardwaru

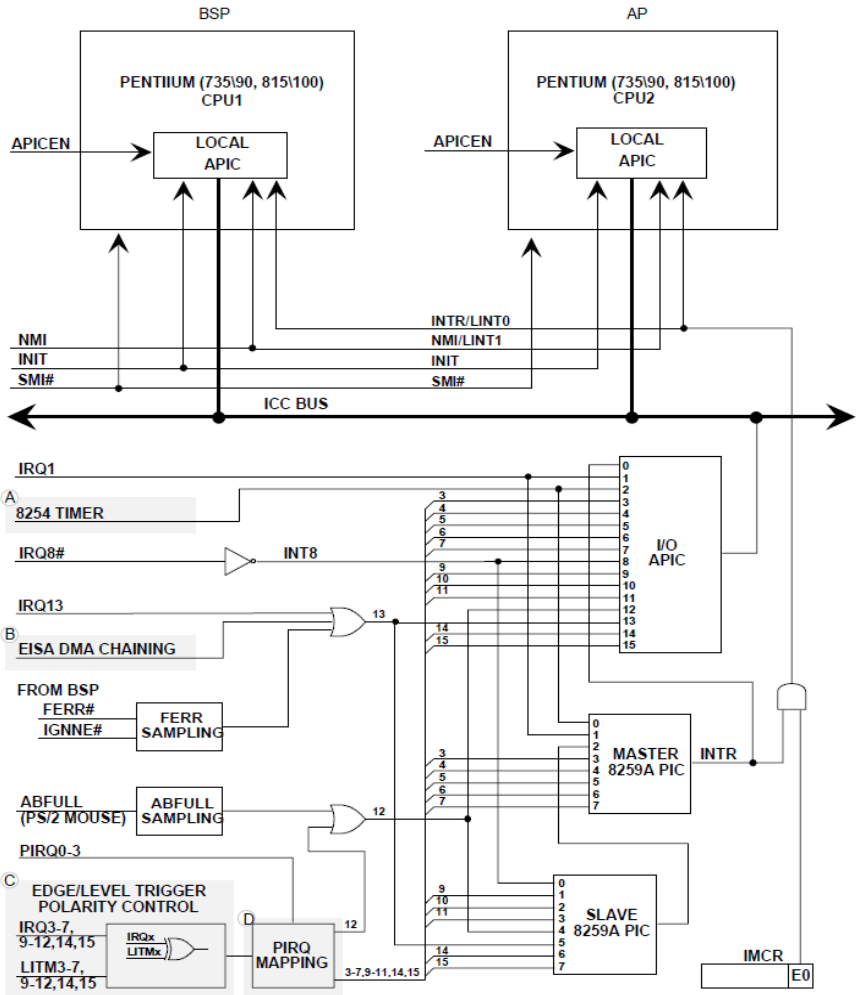
Přerušení dle původu

- Vnější přerušení – zdrojem jsou periferní obvody (časovač, COM port apod.)
- Vnitřní přerušení – výjimky
  - Dělení nulou
  - Výpadek stránky
  - Porušení paměťové ochrany
- Softwarové přerušení
  - Vyvoláno instrukcí INT <číslo>

Přerušení v chráněné režimu

- Adresy obslužných podprogramů jsou umístěny v IDT (interrupt descriptor table)
- IDT má 256 položek velikosti 8 bytů pro přerušení 0-255
- Před povolením jakéhokoliv přerušení musí operační systém vyplnit IDT

Přerušovací systém Pentia



Převzato z MultiProcessor Specification, Intel, verze 1.4, 1997

Programmable Interrupt Controller (PIC)

- Dědictví (legacy) ze starých typů PC
- Původně realizován kaskádním řezením obvodů s označením 8259
- Do řadiče vstupuje osm vnějších přerušení
- Dva řadiče v kaskádě poskytují 15 přerušení
- Provádí
  - výběr přerušení dle priority
  - Generuje signál INT (žádost o přerušení) pro procesor
  - Posílá číslo vybraného přerušení do procesoru ve speciálním cyklu INTACK
- Poskytuje sadu V/V registrů pro svoji konfiguraci na adrese 20h
- Na konci obsluhy přerušení procesor instrukcí OUT 20h, 20h indikuje řadiči konec obsluhy přerušení a řadič vybere další přerušení pro obsluhu (pokud existuje)
- Řadič dovoluje nastavit různé typy přerušení dle potřeby zařízení
  - Úrovňové (žádost je indikována úrovní signálu žádosti)
  - Hranové (žádost je indikována změnou signálu žádosti)

Přehled přerušení (reálný mód, stará PC)

- INT0,1,3-7 vnitřní přerušení, výjimky
- INT2 - nemaskovatelné přerušení
- INT8 (IRQ0) – systémový časovač 8254
- INT9 (IRQ1) – klávesnice
- INT10 (IRQ2) – druhý 8259
- INT11 (IRQ3) – COM1/COM3 (sdílí)
- INT12 (IRQ4) – COM2/COM4 (sdílí)
- INT13 (IRQ5) – SoundCard
- INT14 (IRQ6) – Floppy
- INT15 (IRQ7) – Paralel port
- INT14 (IRQ8) – Real Time Clock
- INT15 (IRQ9) – Reserved
- INT16,17 (IRQ10,11) – Reserved
- INT18 (IRQ12) – PS2 Mouse
- INT19 (IRQ13) – Matematický koprocesor i387
- INT20 (IRQ14) - Harddisk

Advanced Programmable Interrupt Controler (APIC)

- Zaveden pro podporu multiprocesorových systémů
- Local APIC – každý procesor má svůj
- IO APIC – jeden nebo více, ale společných pro všechny procesory
- Local IO APIC posílají do IO APIC požadavky o přerušení
- Poskytuje konfigurovatelné připojení přerušovacích vstupů na jednotlivá přerušení

Local APIC

- 32 bitové konfigurační registry mapované do paměti
- Vniřní čítač pro generování hodin
- Dva vstupy pro přerušení LINT0 a LINT1
- Sběrnici ICC (Interrupt Controller Communication), kterou je napojen na jeden nebo více IO APIC obvodů

IO APIC

- Má 24 IRQ vstupů pro vnější přerušení
- Má 24 položkovou přesměrovávací tabulku (IRQ x -> INT y)
- Programovatelné registry pro konfiguraci IO APIC
- Jednotku pro komunikaci přes ICC
- Přesměrovávací tabulka obsahuje
  - Číslo vektoru přerušení
  - Prioritu
  - Cílový procesor
  - Metodu výběru procesoru

Distribuce žádostí o přerušení do lokálních APIC

- Statická
  - Přímo do procesoru uvedeného v přesměrovávací tabulce
  - Do podmnožiny procesorů
  - Všem (broadcast)
- Dynamická
  - Do procesoru, který zpracovává proces (task) s nejnižší prioritou
  - Priorita se určuje z hodnot v TPR (Task Priority Register) jednotlivých LAPIC (lokální APIC). Hodnotu TPR může operační systém dynamicky měnit podle momentálně běžícího procesu
  - V případě shodné priority (shodné hodnoty v TPR) se rozhoduje na základě arbitrační priority v rozsahu 0-15, která nulována, pokud procesor přijme žádost o přerušení a u ostatních zvětší hodnotu o jedničku.

Meziprocesorová přerušení

- IPI (Inter Processor Interrupt)
- Jeden procesor chce vyvolat přerušení v jiném procesoru (meziprocesorová komunikace)
- Realizuje se zapsáním čísla vektoru přerušení a identifikátor cílového APICu do registru ICR (Interrupt Command Register) svého APICu, který informuje cílový APIC přes ICC (Interrupt Controller Communication)

Výjimky

- 0 – dělení nulou
- 1 – Debug (používá se ke krokování programu)
- 2 – NMI (Non-mascable Interrupt)
- 3 – Breakpoint (používá se k ladění programu)
- 4 –Přetečení (Overflow) u celočíselných aritmetických operací
- 5 –přístup do paměti mimo dané meze (Bounds check)
- 6 – neplatná operace (Invalid opcode)
- 7 – Zařízení nedostupné (týká se SSE, MMX,…)
- 8 – Dvojitá, nezpracovatelná chyba
- 9 – Problém s externím matematickým koprocesorem
- 10 – Přepnutí kontextu do neplatného task segmentu
- 11 – Nepřítomnost segmentu v paměti
- 12 – Překročení limitu zásobníku (přetečení zásobníku)
- 13 – Chyba ochrany paměti (General protection)
- 14 – Výpadek stránky
- 15 – Rezervováno
- 16 – Chyba floating point
- 17 – Chyba zarovnání paměti
- 18 – Chyba procesoru nebo sběrnice (Machine check)
- 19 – SIMD floating point výjimka
- 20-31 Rezervovány pro budoucí použití

Metody připojování periferií

BI-MPP

Přednáška 4

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- USB 3.0
- USB On-The-Go
- Deskriptory zařízení
- USB 4.0
- Konektor USB-C

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- Universal Serial Bus Specification 3.0, Revision 1.0, Listopad 2008 http://www.usb.org/developers/docs/usb_30_spec_092911.zip

Univerzální sériová sběrnice (USB)

- Byla vyvinuta pro připojení periferií jako náhrada za sériový a paralelní port
- Ve verzi USB1.1 rychlostně pokrývala aplikační oblasti do potřeb audio přenosů
- Potřeba přenášení videa a konkurence FireWire si vynutila verzi USB 2.0 s přenosovou rychlostí 480Mb/s
- Nyní ve verzi USB3.0
- Dodatek On-The-Go dovoluje existenci embedded USB host zařízení

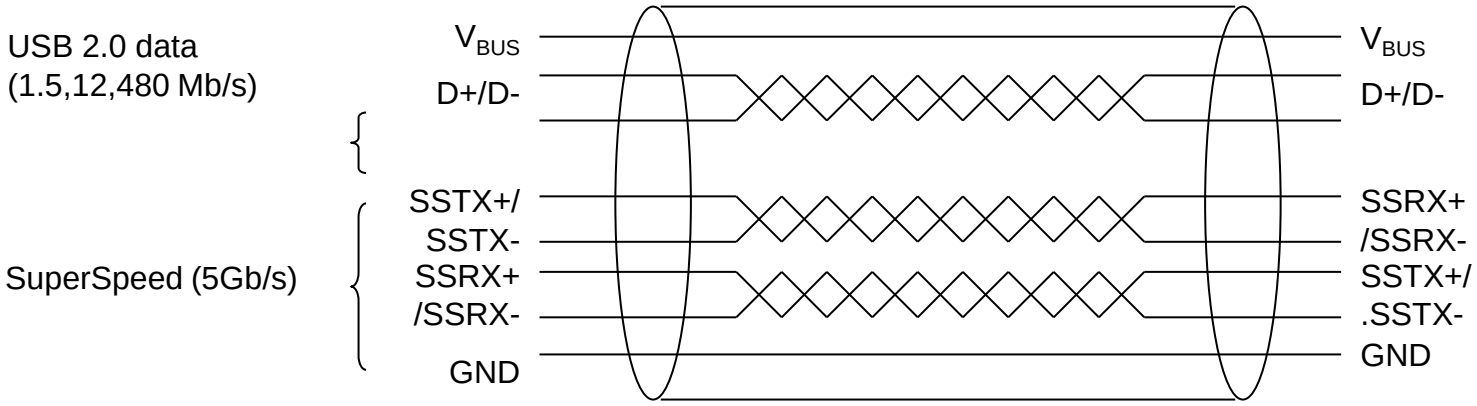
Firmy, které se podílejí na standardu USB

- Hewlett-Packard Company
- Intel Corporation
- Microsoft Corporation
- NEC Corporation
- ST-NXP Wireless
- Texas Instruments

Charakteristika USB

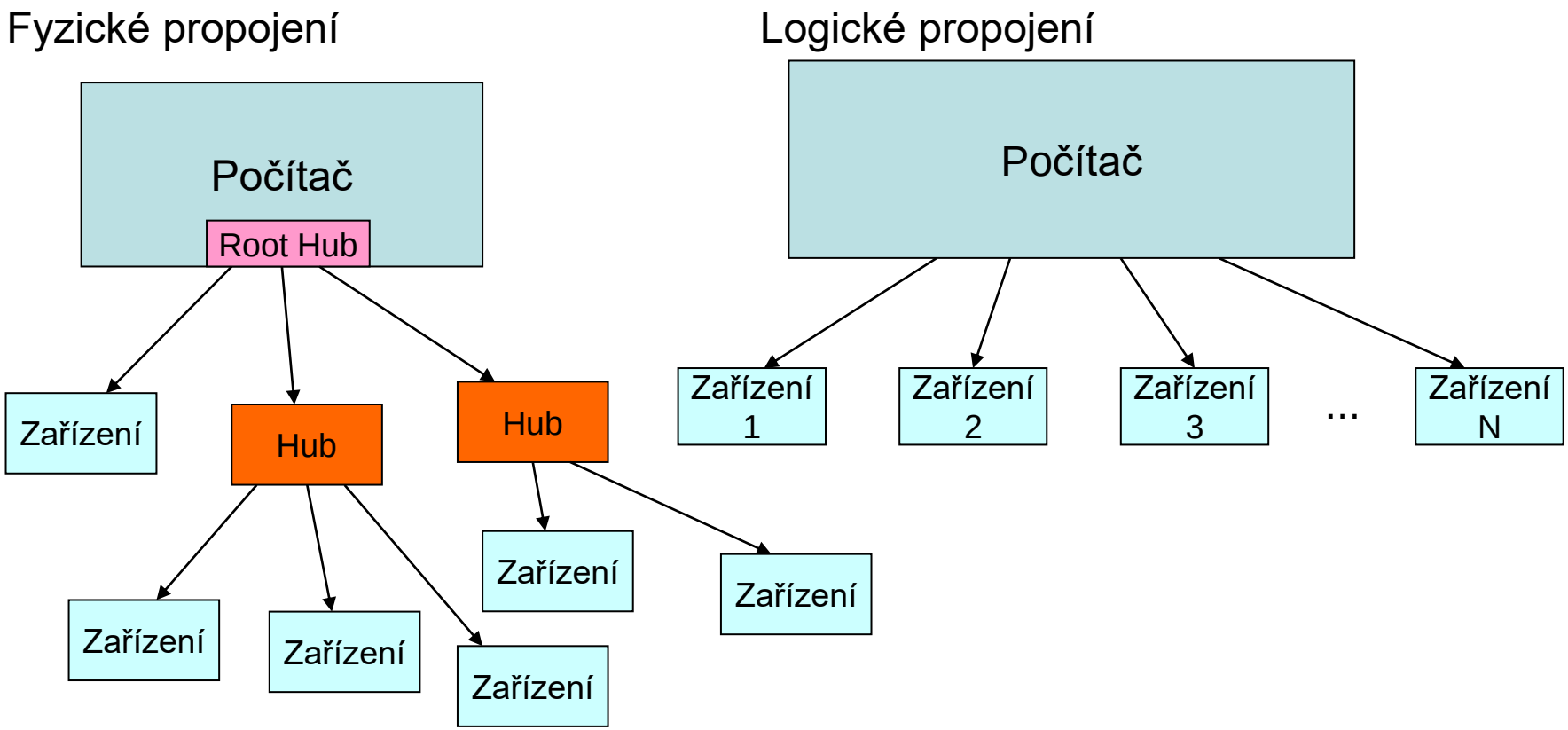
- Fyzické propojení USB zařízení má stromovou strukturu
- Logicky je to sběrnice
- Maximálně lze připojit 127 zařízení
- Maximální vzdálenost pro přímé propojení mezi rozbočovači je daná délkou kabelu a je 5m, maximální vzdálenost od hostitelského počítače je 25m
- Pro USB3.0 jsou definovány přenosové rychlosti 1.5Mb/s, 12Mb/s, 480Mb/s, 5Gb/s (oddělené linky)

Kabel USB 3.0



1	červený	VBUS	Napájení +5V
2/3	bílý/zelený	D-/D+	Datový nestíněný pár USB 2.0
4	černý	GND	Napájení 0V
5/6	modrý/žlutý	SDP1-/SDP1+	Datový stíněný pár SuperSpeed
7			SDP1 stínění
8/9	fialová/oranž.	SDP2-/SDP2+	Datový stíněný pár SuperSpeed
10			SDP2 stínění

Fyzické propojení vs. logické propojení



USB On-The-Go

- Dodatek k USB2.0
- Dovoluje zařízením stát se hostitelským počítačem
- Protokol podporuje vzájemnou dohodu zařízení, kdo je hostitel a kdo zařízení
- Dovoluje například kopírovat data z fotoaparátu na USB disk bez nutnosti počítače, tisk fotografií přímo z USB flashky, atd.
- Dnes je On-The-Go řadič běžnou součástí některých typů mikropočítačů a embedded procesorů

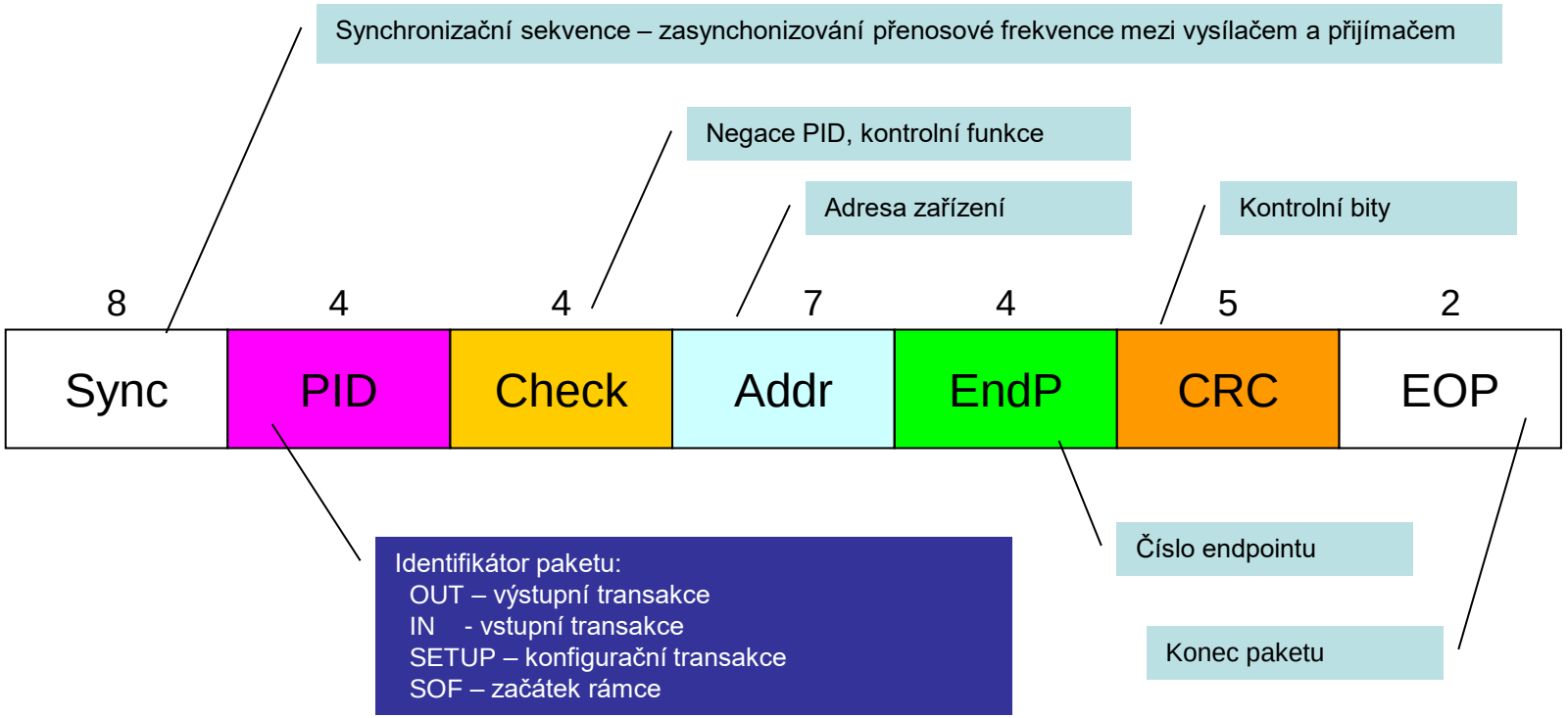
Endpointy

- Endpoint je koncový bod na USB zařízení, se kterým hostitelský počítač komunikuje
- Endpoint je fyzicky hardwarově realizován a typicky obsahuje frontu pro příjem dat
- Každé zařízení má povinný Endpoint 0, který se používá pro konfiguraci zařízení, ostatní endpointy tvoří rozhraní logických zařízení
- Maximální počet Endpointů je 16
- Každý Endpoint má určen směr přenosu
- Všechny Endpointy sdílejí společný komunikační kanál (USB kabel), číslo Endpointu je součástí komunikačního protokolu.

Přenosy po USB

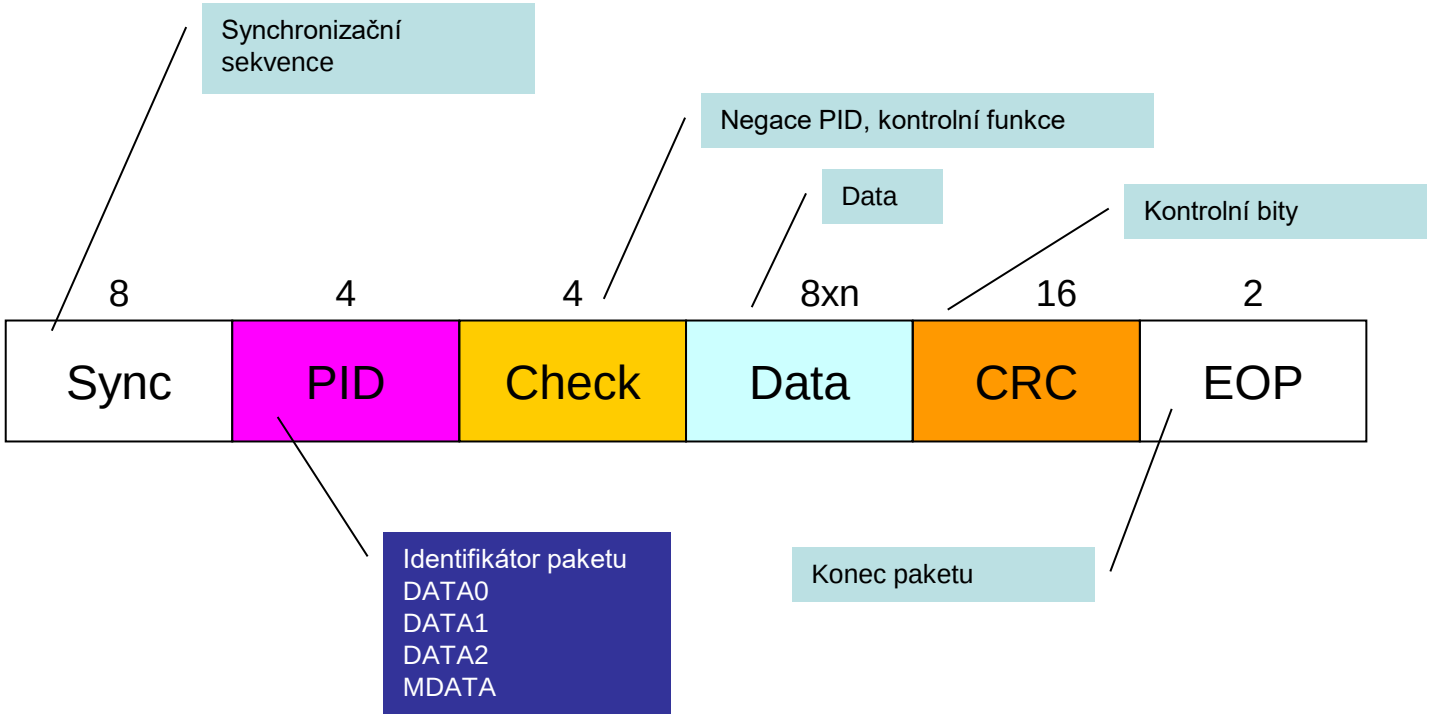
- Rozeznáváme přenosy
  - blokové (spolehlivé, opakování přenosu při chybě)
  - isochronní (při chybě se informace ztrácí)
  - řídící
- Přenosy se skládají z transakcí, kterou tvoří
  - Hlavičkový paket (token), který nese informaci o typu přenosu (IN/OUT/SETUP), adresu zařízení a endpoint
  - Datový paket (DATA0,DATA1, …)
  - Potvrzení (ACK, NACK, STALL, nebo bez potvrzení)

Formát hlavičkového paketu (token)

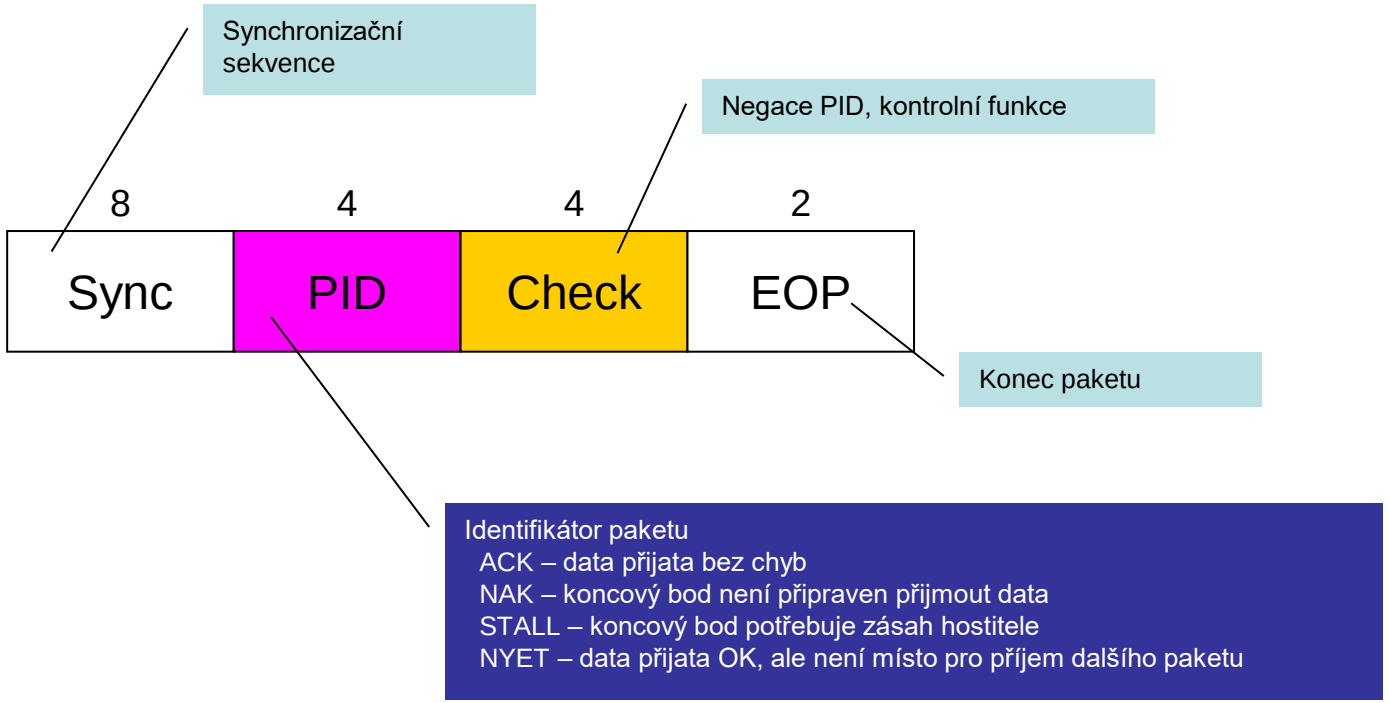


Formát přenosů určuje řadu limitů: počet typů rámce, počet adresovatelných zařízení a počet endpointů.

Formát datového paketu

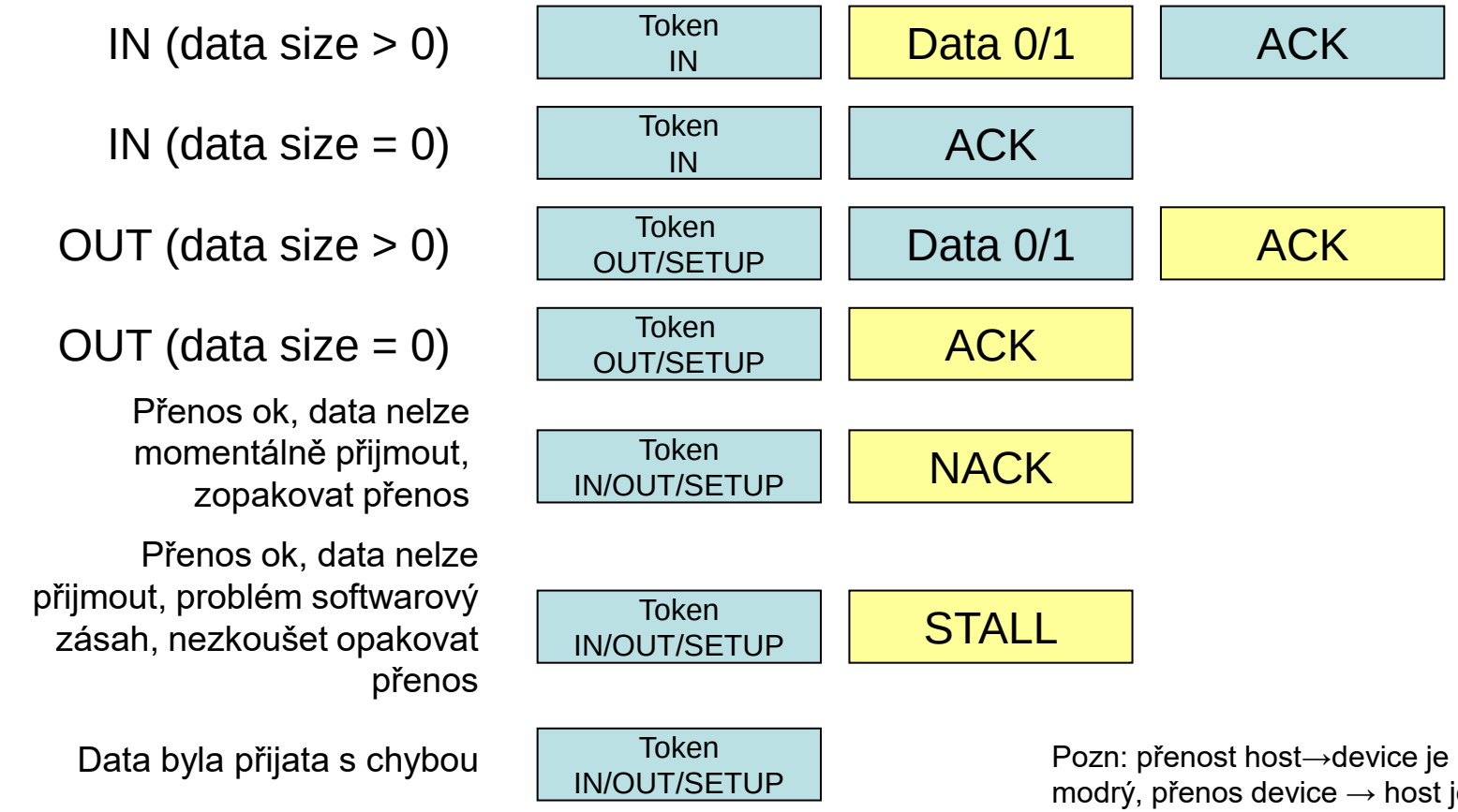


Formát potvrzovacího paketu



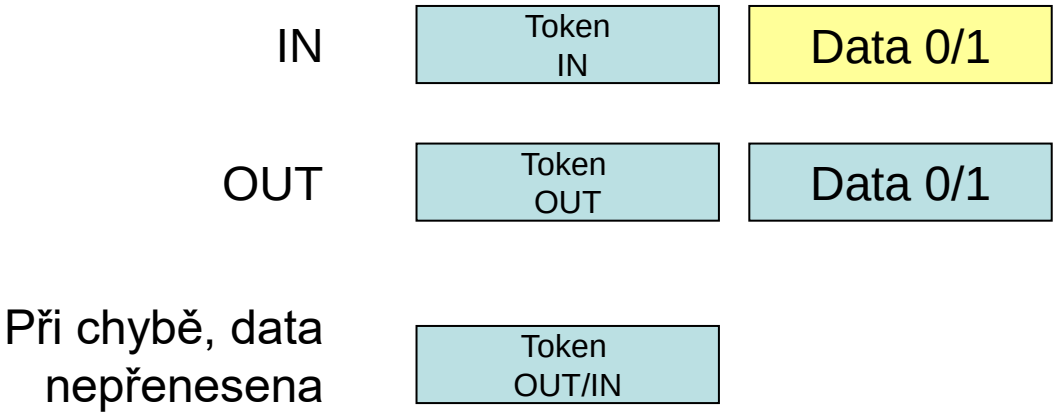
Blokové transakce (Bulk transactions)

Blokové transakce jsou základními přenosovými jednotkami pro blokové přenosy (bulk transfers)



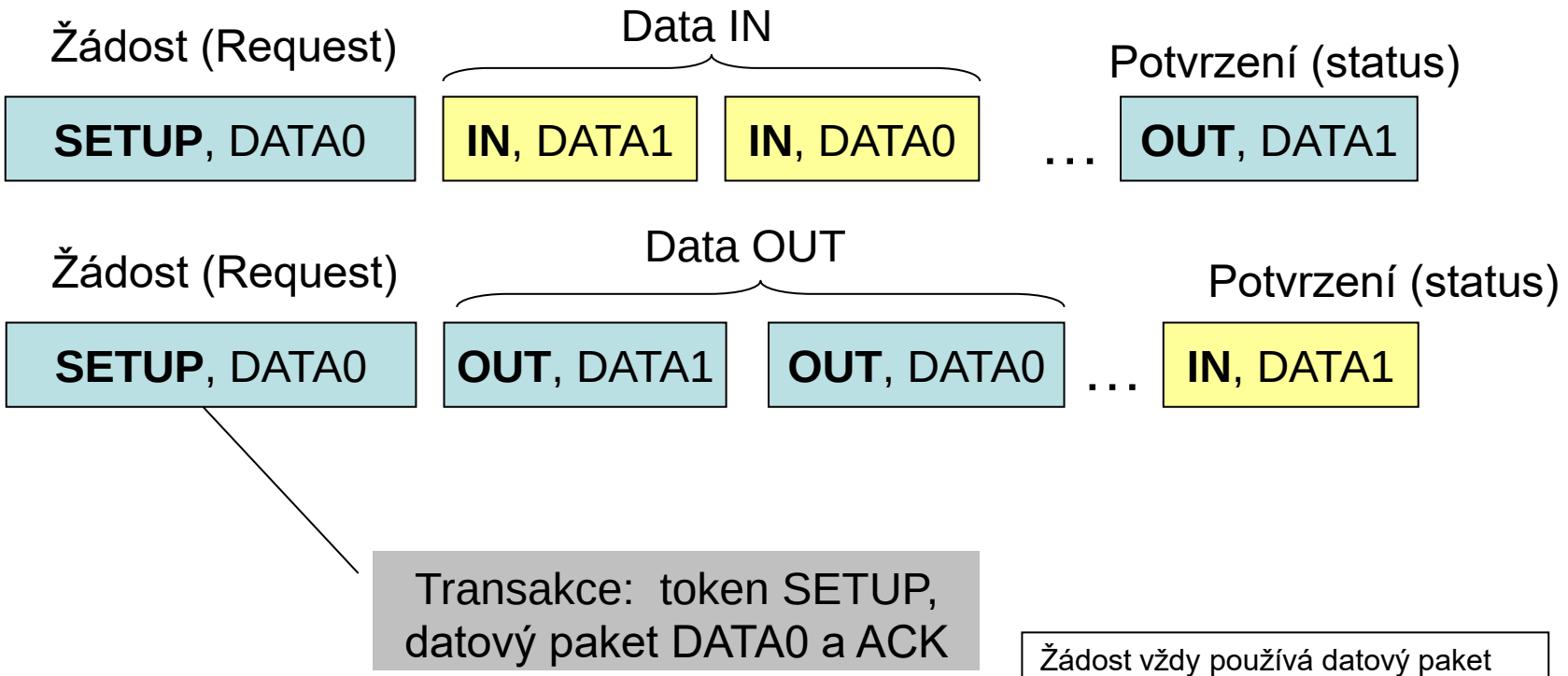
Pozn: přenost host→device je modrý, přenos device → host je žlutý

Isochronní transakce



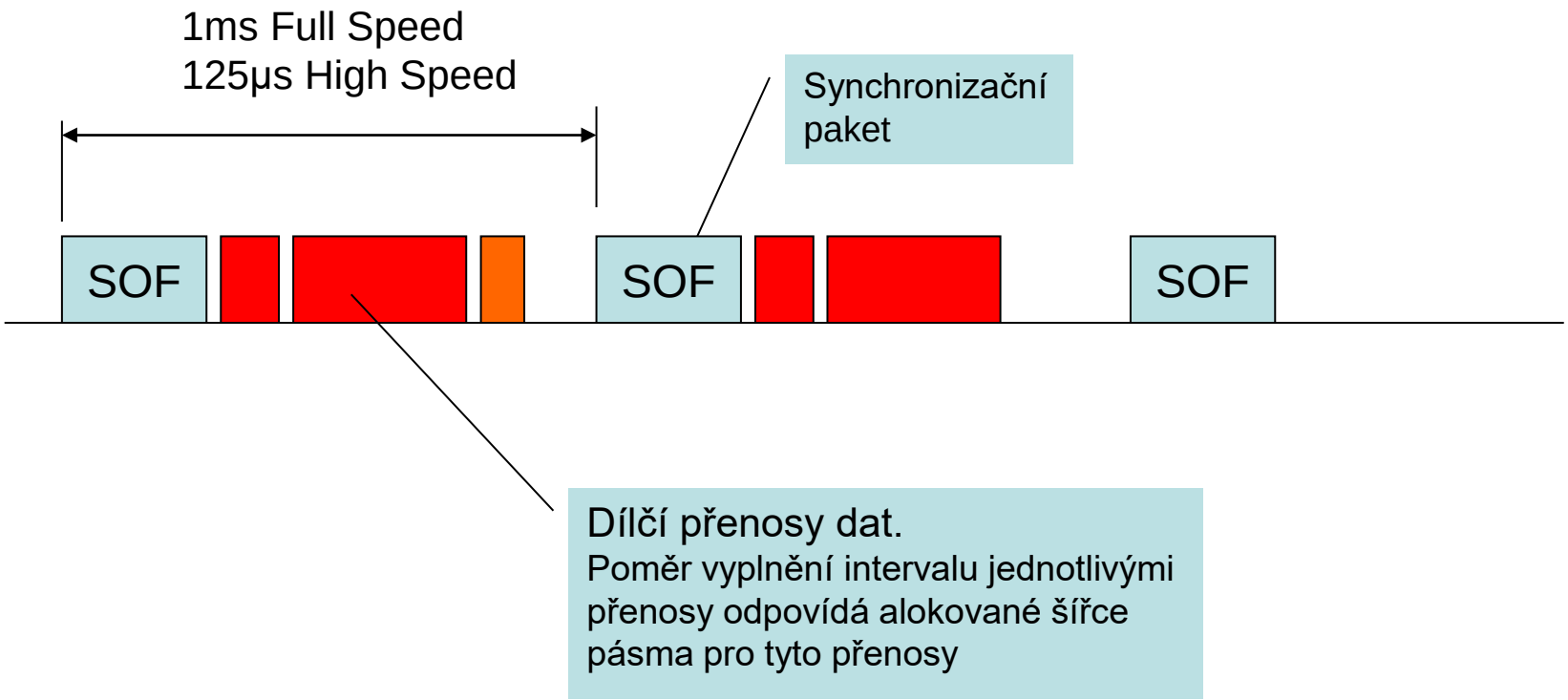
Isochronní transakce se od blokových liší tím, že nemají potvrzování a data nelze při chybě nebo nemožnožnosti přijetí opakovat

Řídící přenosy



Žádost vždy používá datový paket DATA0, pak se datové pakety DATA0,DATA1 střídají a potvrzení používá vždy datový paket DATA1.

Synchronizace přenosů (rámce)

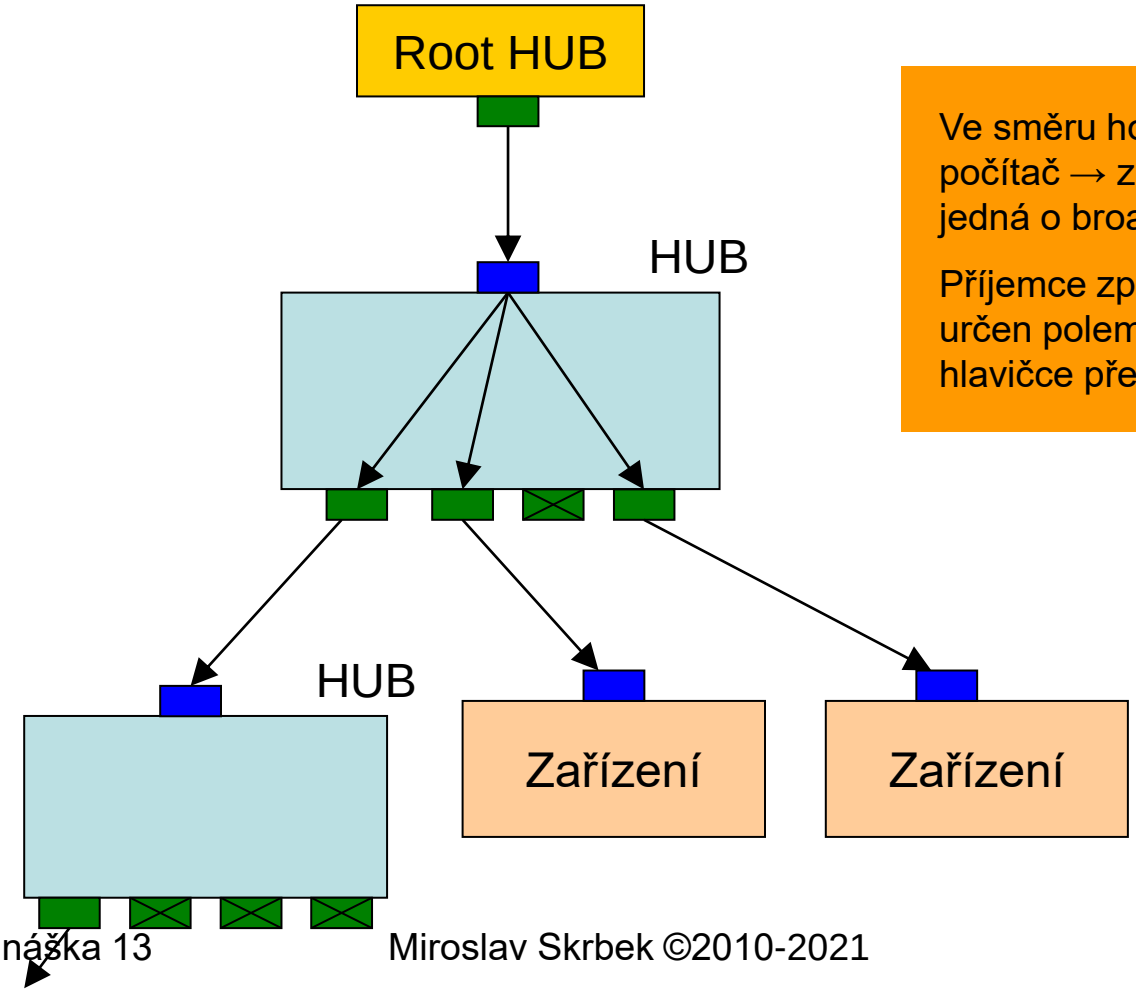


Typy přenosů

- Řídící přenosy – slouží ke konfiguraci zařízení a vyčítání PnP informace ze zařízení. Doručení dat zaručeno.
- Hromadné přenosy – přenos dat bez požadavku na termín a rychlost doručení. Doručení dat je garantováno. V případě chyby se přenos dat opakuje. Mají nejnižší prioritu
- Izochronní přenosy – přenosy dat v reálném čase. Určeno pro přesnost audio a video dat. Není zaručeno dodání dat (možnost ztráty, při chybě se přenosy neopakují). V rámci maximální šířky pásma si mohou jednotlivé izochronní kanály alokovat požadovanou šířku pásma (přenosovou rychlost).
- Přerušení – přenosy dat od zařízení s minimálním zpožděním. Požadavky na tyto přenosy vznikají asynchronně na straně zařízení, avšak do hostitelského počítače se přenášejí až na žádost hostitelského počítače (polling). Perioda dotazů na interrupt endpointy je popsána v deskriptoru pro tento endpoint.

Propagace dat hierarchií HUBů

směr hostitelský počítač → zařízení

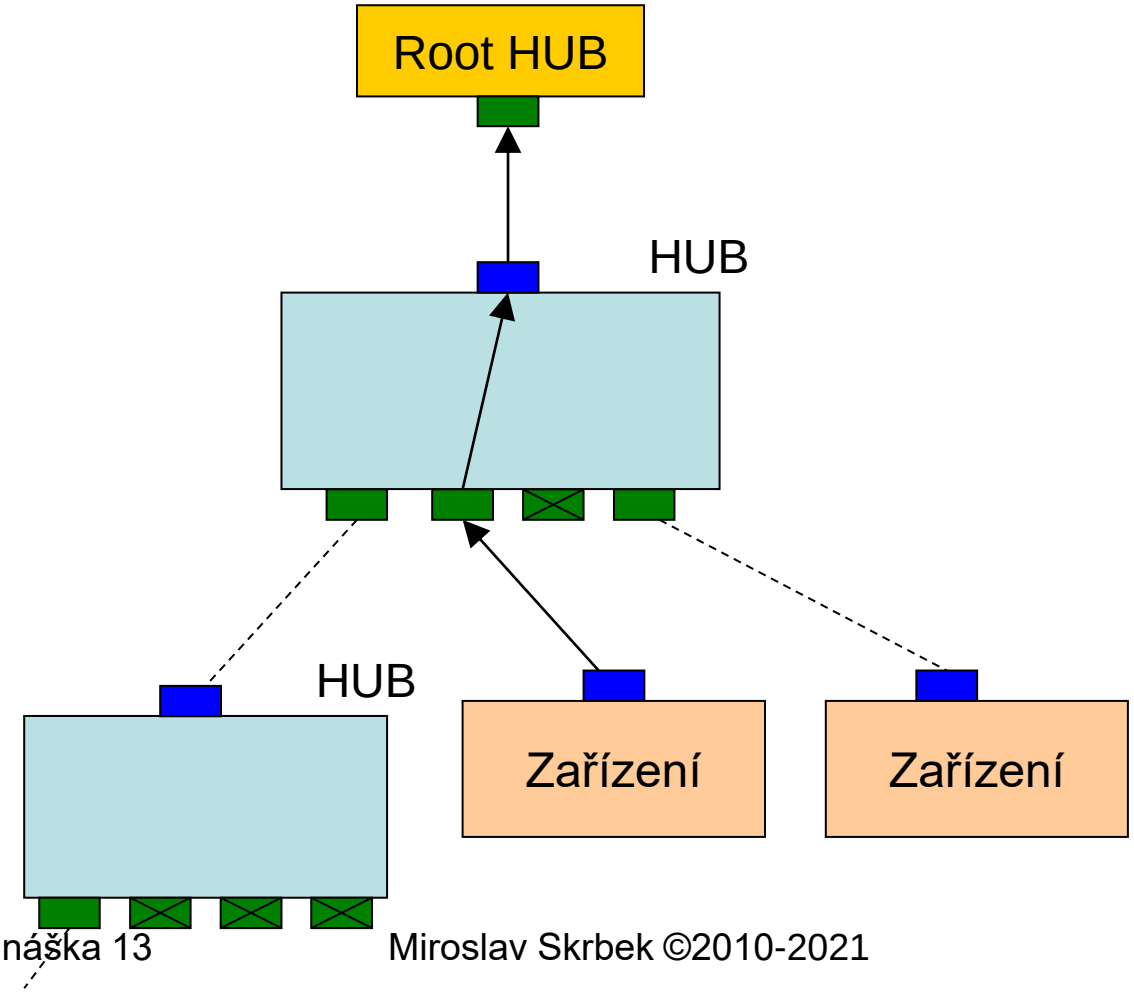


Ve směru hostitelský počítač → zařízení se jedná o broadcast.

Příjemce zprávy je určen polem addr v hlavičce přenosu.

Propagace dat hierarchií HUBů

směr zařízení → hostitelský počítač



USB a PnP

- USB zařízení jsou typu PnP (plug-and-play)
- PnP informace o zařízení je uložena v deskriptorech a lze ji ze zařízení vyčíst speciálními konfiguračními požadavky (Requests)
- Na základě PnP informací operační systém instaluje ovladače pro dané zařízení

Deskriptory USB zařízení

- Identifikují zařízení
  - Jednoznačně VID a PID
  - Jako zařízení dané třídy
- Popisují zařízení z hlediska
  - Konfigurací (configuration)
  - Rozhraní (interface)
  - Endpointů
- Čtou se přes endpoint 0
  - Bulk přenosem typu SETUP
- Request GetDescriptor

Typy deskriptorů

- DeviceDescriptor
- Configuration Descriptor
- Interface Descriptor
- EndPoint Descriptor
- Speciální typy
  - Popisující HUB
  - Popisující HID zařízení

DeviceDescriptor

```
typedef struct device_descr
{
    byte bLength;
    byte bDescriptorType;     // povinné pro všechny deskriptory
    word bcdUSB;              // verze USB
    byte bDeviceClass;        // třída zařízení
    byte bDeviceSubClass;     // podtřída zařízení
    byte bDeviceProtocol;     // typ protokolu
    byte bMaxPacketSize;      // velikost bufferu pro SETUP přenosy
    word idVendor;            // identifikátor výrobce
    word idProduct;           // identifikátor produktu
    word bcdDevice;
    byte iManufacturer;       // index string deskriptoru výrobce
    byte iProduct;            // index string descriptoru produktu
    byte iSerialNumber;       // index string descriptoru ser. cisla
    byte bNumConfigurations;  // počet konfigurací zařízení
} device_descr_t;
```

VID a PID

- VID (Vendor ID) je číslo, které jednoznačně identifikuje výrobce
- PID (Produkt ID) je číslo, které jednoznačně identifikuje produkt daného výrobce
- VID a PID se užívá pro vyhledání driveru pro konkrétní zařízení

Class a SubClass zařízení

- Class a SubClass jsou čísla, která určují druh zařízení (např. MassStorage Device, HID Device, apod.)
- Zařízení, které udává třídu a podtřídu musí být s třídou těchto zařízení kompatibilní (standardizováno)
- Pro zařízení v dané třídě fungují univerzální třídní ovladače a není potřeba ovladač pro konkrétní zařízení.
- Typickým USB zařízením užívajícím třídní ovladače je MassStorage Device (USB klíčenka)

Indexy string deskriptorů

- iManufacturer, iProduct a iSerialNumber jsou indexy (čísla) string deskriptorů.
- String deskriptory obsahují textovou informaci v jednom nebo více jazycích popisující výrobce, produkt a sériové číslo.
- Obsahy deskriptorů se zobrazují v os windows ve žluté bublině při zasunutí zařízení.

Configuration Descriptor

```
typedef struct config_descriptor {
    byte bLength;
    byte bDescriptorType;
    word wTotalLength;                  // celková délka včetně
                                        // interface deskriptorů,
                                        // které následují
    byte bNumInterface;                 // počet interface deskriptorů
    byte bConfigurationValue;           // konfigurační hodnota
    byte iConfiguration;                // index string deskr. Popisujici
                                        // konfiguraci
    byte bmAttributes;                  // atributy konfigurace
    byte bMaxPower;                     // maximální odběr zařízení v dané
                                        // konfiguraci v mA
} config_descriptor_t;
```

Konfigurační deskriptor

- Popisuje konfiguraci zařízení. Různé konfigurace se mohou lišit ve spotřebě, zapnutí určitých funkcionalit zařízení, různé velikosti bufferů na interfacech apod.
- Konfiguračních deskriptorů může být více, podle počtu možných konfigurací
- V daném okamžiku je aktivní pouze jedna konfigurace a vybírá se USB requestem na základě hodnoty v položce bConfigurationValue.

Interface Descriptor a Endpoint Descriptor

```
typedef struct interf_descriptor {
    byte bLength;
    byte bDescriptorType;
    byte bInterfaceNumber;
    byte bAlternateSetting;
    byte bNumEndpoints;
    byte bInterfaceClass;
    byte bInterfaceSubClass;
    byte bInterfaceProtocol;
    byte iInterface;
} interf_descriptor_t;
```

```
typedef struct endpoint_descriptor {
    byte bLength;
    byte bDescriptorType;
    byte bEndpointAddress;
    byte bmAttributes;
    word wMaxPacketSize;
    byte bInterval;
} endpoint_descriptor_t;
```

USB4

- USB-C konektor (5V/3A)
- Povinná specifikace USB Power Delivery
- Přenosová rychlost až 40Gb/s
- Tunelování
  - USB3.2
  - Display portu
  - PCI Express
- USB-C konektor obsahuje USB2 D+,D-

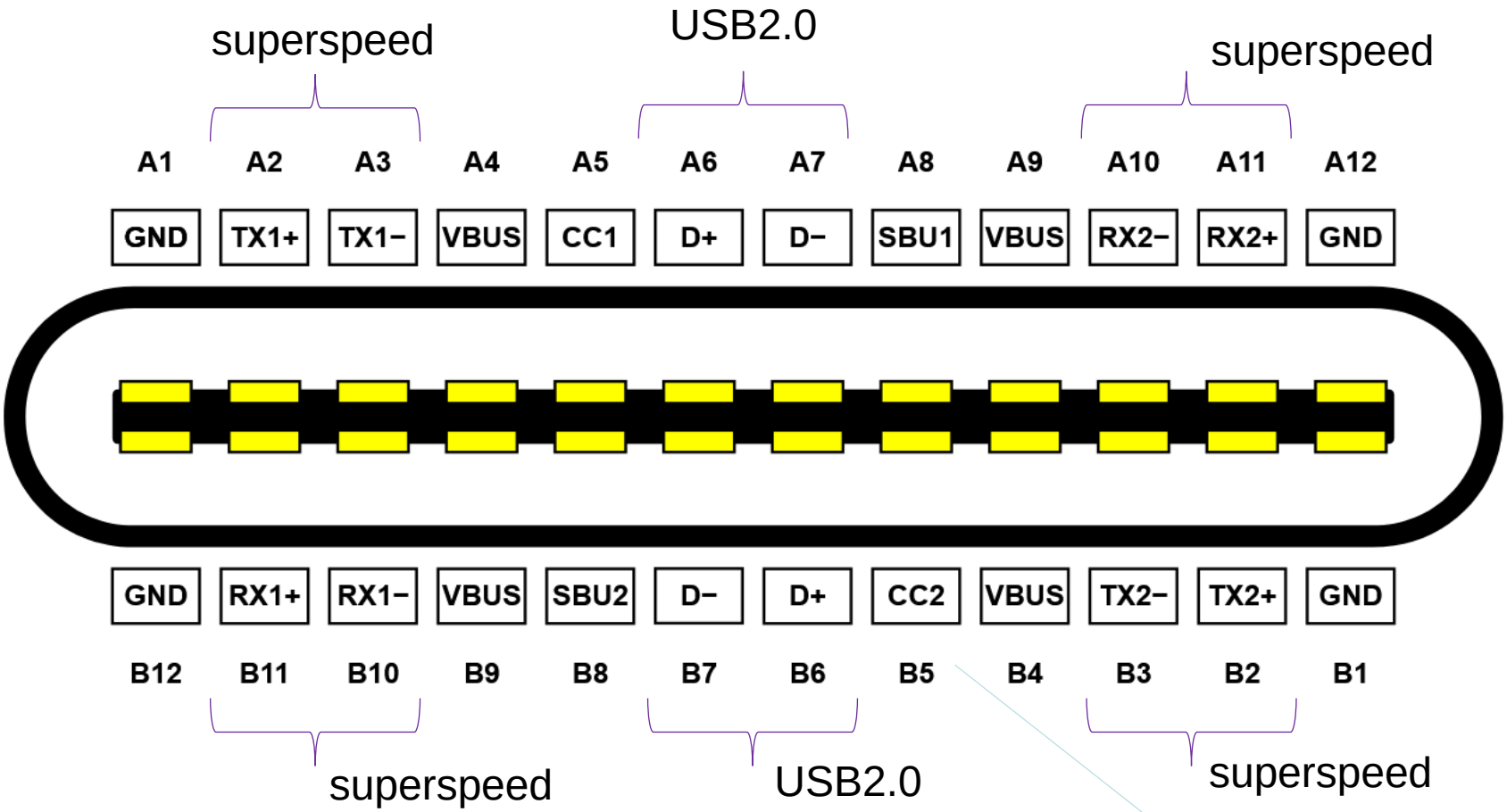
Komunikační rychlosti USB

- Low Speed - 1.5Mb/s, (USB1.0,1.1,2.0)
- Full Speed - 12Mb/s, (USB1.0,1.1,2.0)
- High Speed - 480Mb/s (USB2.0)
- Super Speed - 5Gb/s (USB3.0)
- Super Speed+ - 10Gb/s (USB3.1)
- Super Speed+ - 20Gb/s (USB3.2)
- SuperSpeed+ - 40Gb/s (USB4)

USB konektory

- Typ A (host) USB1.0,1.1,2.0
- Typ B (device) USB 1.0,1.1,2.0
- Typ C (host,device) USB2.0,3.0,3.1,3.2,4
- Mini/micro verze
  - Mini A, mini B (USB1.1,2.0)
  - Mini AB USB2.0
  - Micro A, micro AB (USB2.0)
  - Micro B (USB3.0)

USB-C konektor

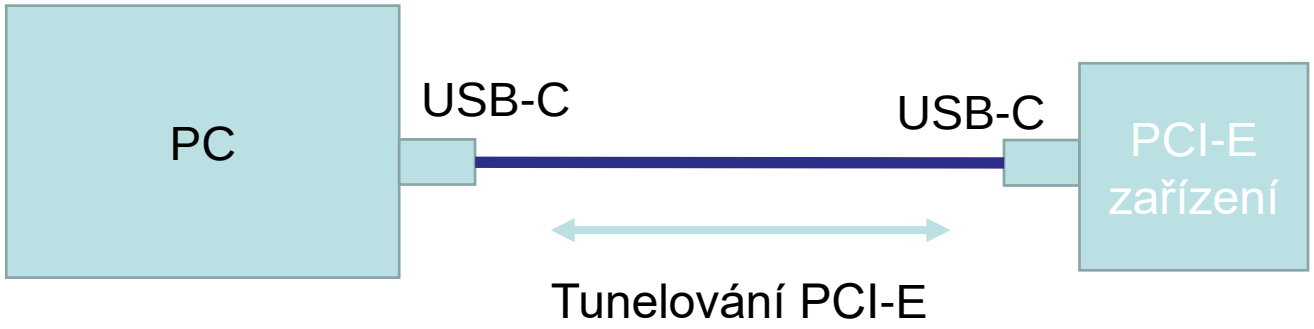


Zdroj:https://en.wikipedia.org/wiki/USB#/media/File:USB_Type-C_Receptacle_Pinout.svg

Co znamená tunelování PCI Express ?

Uvědomíme si, že PCI-E je systémová sběrnice, proto registry periferie jsou mapovány do paměťového nebo IO prostoru počítače PC a přistupujeme k nim pomocí paměťových operací a IO operací.

Zařízení připojené přes „USB“ detekujeme na Linuxu příkazem lspci.



Metody připojování periferií

BI-MPP

Přednáška 5

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- SCSI
- Zařízení typu Mass Storage
- SCSI příkazy

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- Universal Serial Bus Specification 3.0, Revision 1.0, Listopad 2008 http://www.usb.org/developers/docs/usb_30_spec_092911.zip
- Universal Serial Bus Mass Storage Class Bulk-Only Transport Revision 1.0 September 31, 1999 (http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf)

SCSI

- Protokol SCSI využívá řada sběrnic pro přístup k zařízením typu disk, CDROM apod.
- Dnes nejvýznamnější příklady
  - Firewire
  - USB Mass Storage
- Transportní protokoly poskytují wrappery, které obalují SCSI příkazy

USB Mass Storage

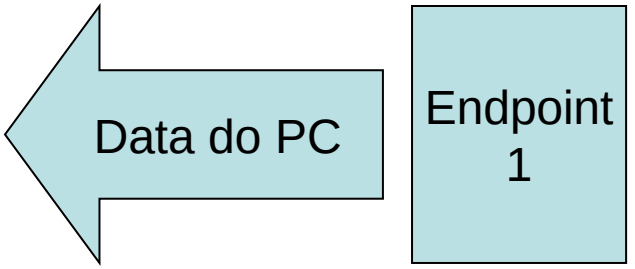
Device deskriptor

```
Device Descriptor
  bLength                18
  bDescriptorType         1
  bcdUSB               2.00
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0        64
  idVendor           0x13fe Kingston Technology Company Inc.
  idProduct          0x1d00 DataTraveler 2.0 1GB/4GB Flash
                            Drive / Patriot Xporter 4GB Flash
                              Drive
  bcdDevice            1.10
  iManufacturer           1 Kingston
  iProduct                2 DataTraveler 2.0
  iSerial                 3 5B720D9BA39C
```

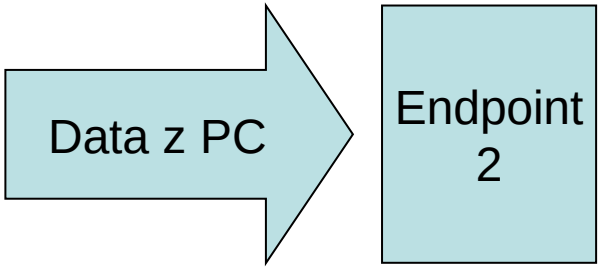
USB Mass Storage

Endpoint deskriptory

```
Endpoint Descriptor:
  bLength             7
  bDescriptorType     5
  bEndpointAddress    0x81  EP 1 IN
  bmAttributes        2
  Transfer Type       Bulk
  Synch Type          None
  Usage Type          Data
  wMaxPacketSize      0x0200  1x 512 bytes
  bInterval           0
```




```
Endpoint Descriptor:
  bLength             7
  bDescriptorType     5
  bEndpointAddress    0x02  EP 2 OUT
  bmAttributes        2
  Transfer Type       Bulk
  Synch Type          None
  Usage Type          Data
  wMaxPacketSize      0x0200 1x 512 bytes
  bInterval           0
```



Zjištění maximálního počtu jednotek

SCSI zařízení je děleno na logické jednotky (nezávislá zařízení).

Logická zařízení jsou označována čísly LUN (Logical Unit Number).

USB Mass Storage zařízení poskytují USB request na endpointu 0 pro zjištění maximálního čísla LUN implementované jednotky.

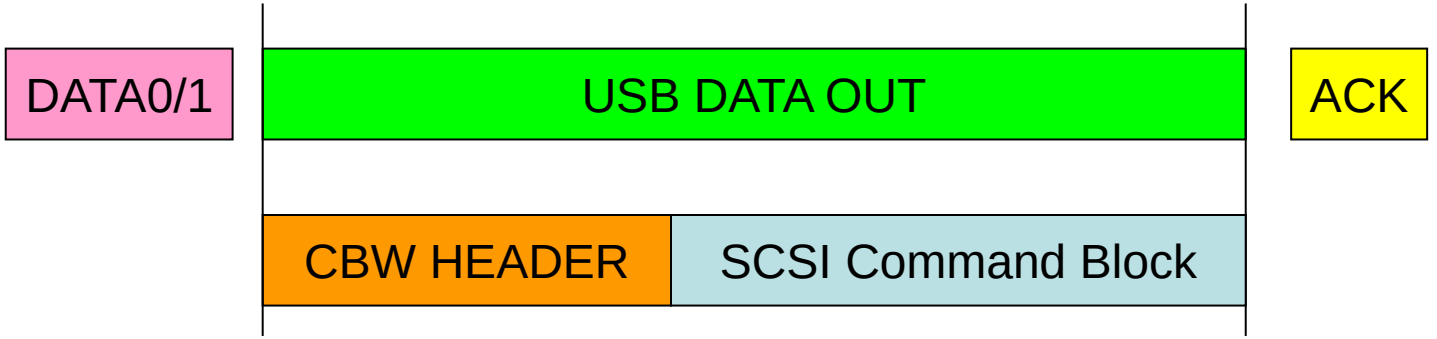
Číslování je od nuly: 0, 1, 2, …, MAX LUN;
počet jednotek je MAX LUN + 1

Get MAX LUN

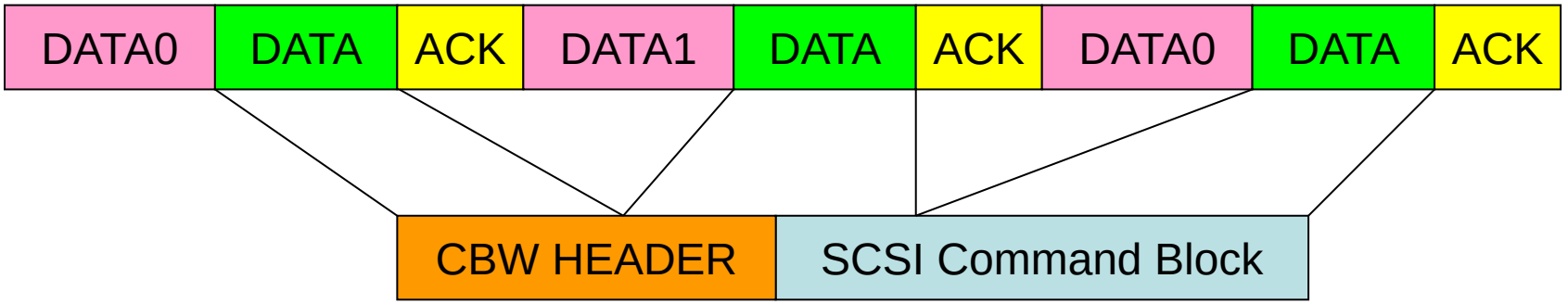
bmRequestType	bRequest	wValue	wIndex	wLength	Data
A1h	FEh	0	Interface	1h	1 byte

Zapouzdření SCSI transportního protokolu do USB přenosů

Varianta 1



Varianta 2



Control Block Wrapper (CBW)

```
typedef struct __attribute__((packed)) {
        unsigned long dCBWSignature;
        unsigned long dCBWTag;
        unsigned long dCBWDataTransferLength;
        unsigned char bmCBWFlags;
        unsigned char bCBWLUN;
        unsigned char bCBWCBLength;
        unsigned char CBWCB[16];
 } CBW_t;
```

= 43425355h

Délka přenosu

LUN jednotka, pro kterou je CWB určen

SCSI Command Block

Délka sekce CBWCB (SCSI Command Block)

dCBWTag – číslo, které páruje CBW s CSW (Control Status Wrapper)

Command Status Wrapper (CSW)

```
typedef struct __attribute__((packed)) {
        unsigned long dCSWSignature;
        unsigned long dCSWTag;
        unsigned long dCSWDataResidue;
        unsigned char bCSWStatus;
} CSW_t;
```

=53425355h

SCSI status

Zbývající nepřenesená délka dat

Status uzavírá každou transakci

Zjištění typu zařízení (INQUIRY)

SCSI Command Block

```
CBW.CBWCB[0]=0x12
CBW.CBWCB[1]=0
CBW.CBWCB[2]=0
CBW.CBWCB[3]=alokacni_delka_msb
CBW.CBWCB[4]=alokacni_delka_lsb
CBW.CBWCB[5]=0
```

Typická délka dat zaslaných zařízením je 36 bytů

Zjištění stavu zařízení (REQUEST SENSE)

SCSI Command Block

```
CBW.CBWCB[0]=0x03
CBW.CBWCB[1]=0
CBW.CBWCB[2]=0
CBW.CBWCB[3]=0
CBW.CBWCB[4]=alokacni_delka // = 18 bytů
CBW.CBWCB[5]=0
```

Typická délka dat zaslaných zařízením je 18 bytů

Zjištění kapacity disku READ CAPACITY

SCSI Command Block

```
CBW.CBWCB[0]=0x25
CBW.CBWCB[1]=0
CBW.CBWCB[2]=0
CBW.CBWCB[3]=0
CBW.CBWCB[4]=0
CBW.CBWCB[5]=0
CBW.CBWCB[6]=0
CBW.CBWCB[7]=0
CBW.CBWCB[8]=0
CBW.CBWCB[9]=0
```

Data – odpověď na READ CAPACITY

```
data[0] … LBA_MAX_MSB
data[1] … LBA_MAX_byte2
data[2] … LBA_MAX_byte1
data[3] … LBA_MAX_LSB
data[4] … BLOCK_LEN_MSB
data[5] … BLOCK_LEN_byte2
data[6] … BLOCK_LEN_byte1
data[7] … BLOCK_LEN_LSB
```

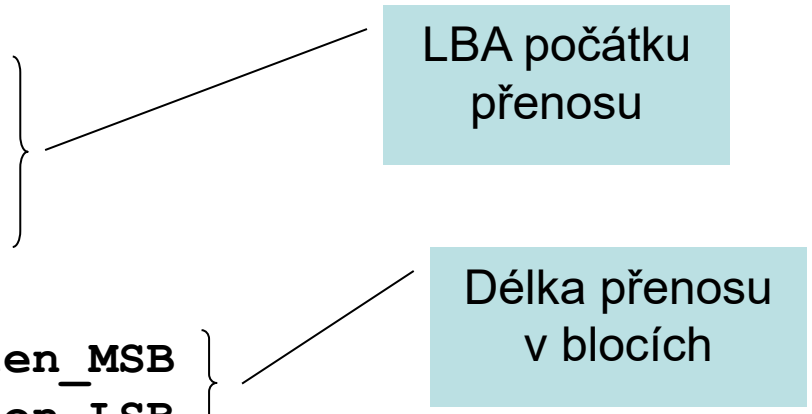
Maximální LBA adresa

Velikost bloku v bytech

CAPACITY = (LBA_MAX +1) * BLOCK_LEN (v bytech)

Čtení bloku READ

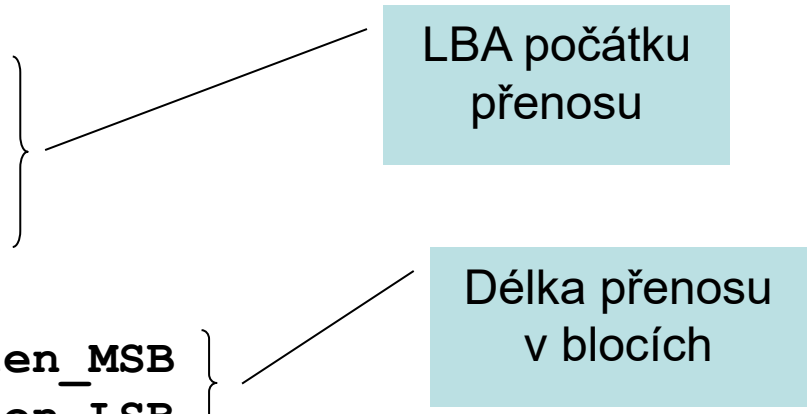
```
CBW.CBWCB[0]=0x28
CBW.CBWCB[1]=0
CBW.CBWCB[2]=LBA_MSB
CBW.CBWCB[3]=LBA_byte2
CBW.CBWCB[4]=LBA_byte1
CBW.CBWCB[5]=LBA_LSB
CBW.CBWCB[6]=0
CBW.CBWCB[7]=transfer_len_MSB
CBW.CBWCB[8]=transfer_len_LSB
CBW.CBWCB[9]=0
```



Tento SCSI Command Block následují požadovaná data.

Zápis bloku WRITE

```
CBW.CBWCB[0]=0x2A
CBW.CBWCB[1]=0
CBW.CBWCB[2]=LBA_MSB
CBW.CBWCB[3]=LBA_byte2
CBW.CBWCB[4]=LBA_byte1
CBW.CBWCB[5]=LBA_LSB
CBW.CBWCB[6]=0
CBW.CBWCB[7]=transfer_len_MSB
CBW.CBWCB[8]=transfer_len_LSB
CBW.CBWCB[9]=0
```



Tento SCSI Command Block následují požadovaná data.

Metody připojování periferií

BI-MPP

Přednáška 6

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- ATA
- SATA
- SATA AHCI
- IEEE1394 FireWire
- LPC

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- IEEE Standard for a High-Performance Serial Bus. IEEE Computer Society
- Serial Bus Protocol 2 (SBP-2). T10 Project 1155D, Revision 4, May 19, 1998
- Intel® Low Pin Count (LPC) Interface Specification. Intel Corporation. Revision 1.1, August 2002

ATA

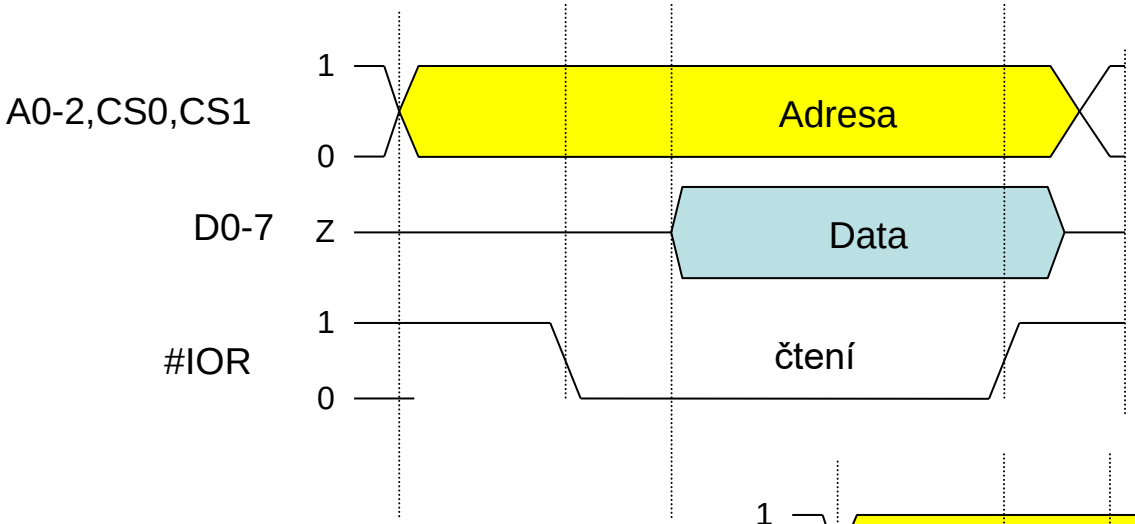
- Paralelní rozhraní disků
- Vychází ze sběrnice ISA (podmnožina signálů, stejné časování)
- Postupně zrychlen přenos dat při DMA
- Signály
  - D0-15 data
  - A0-2 adresa
  - CS0 a CS1 výběr rozsahu adres pro blok příkazových registrů (1F0h-1FFh) a blok řídících registrů (3F0h-3FFh)
  - IOW#, IOR# - řídící signály pro zápis a čtení
  - INTRQ – žádost o přerušení
  - DMARQ – žádost o DMA
  - DMACK – potvrzení žádosti o přerušení
  - DASP – indikace přítomnosti disku slave
  - CSEL – adresace disku master/slave
  - PDIAG – indikace dokončení interní diagnostiky
  - RESET# - reset disku (= RESET počítače)
  - IOCS16 – řídící signál indikující 16-bitové přenosy dat
  - IORDY – indikace připravenosti dat

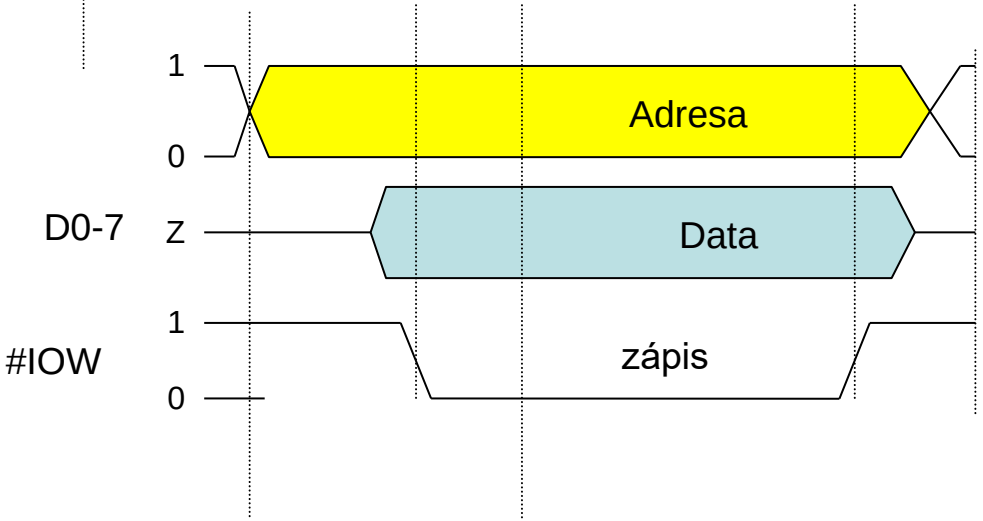
Příklad:Ovládání disku (PIO mód)

Postup:

1. Bit D nastavíme dle toho, zda chceme pracovat s diskem master nebo slave. S tímto je nutno začít !
2. Zapíšeme počet sektorů, které chceme číst do SC
3. Zapíšeme adresu sektoru do všech LBA polí a nastavíme bit L na jedničku (LBA mód). Bit D musí zůstat nezměněn.
4. Zapíšeme příkaz do registru Příkaz
5. Opakovaně čteme registr stav a testujeme bit 7 dokud není nulový.
6. V cyklu šestnáctibitovým čtením přečteme obsah sektoru (256 čtecích cyklů)

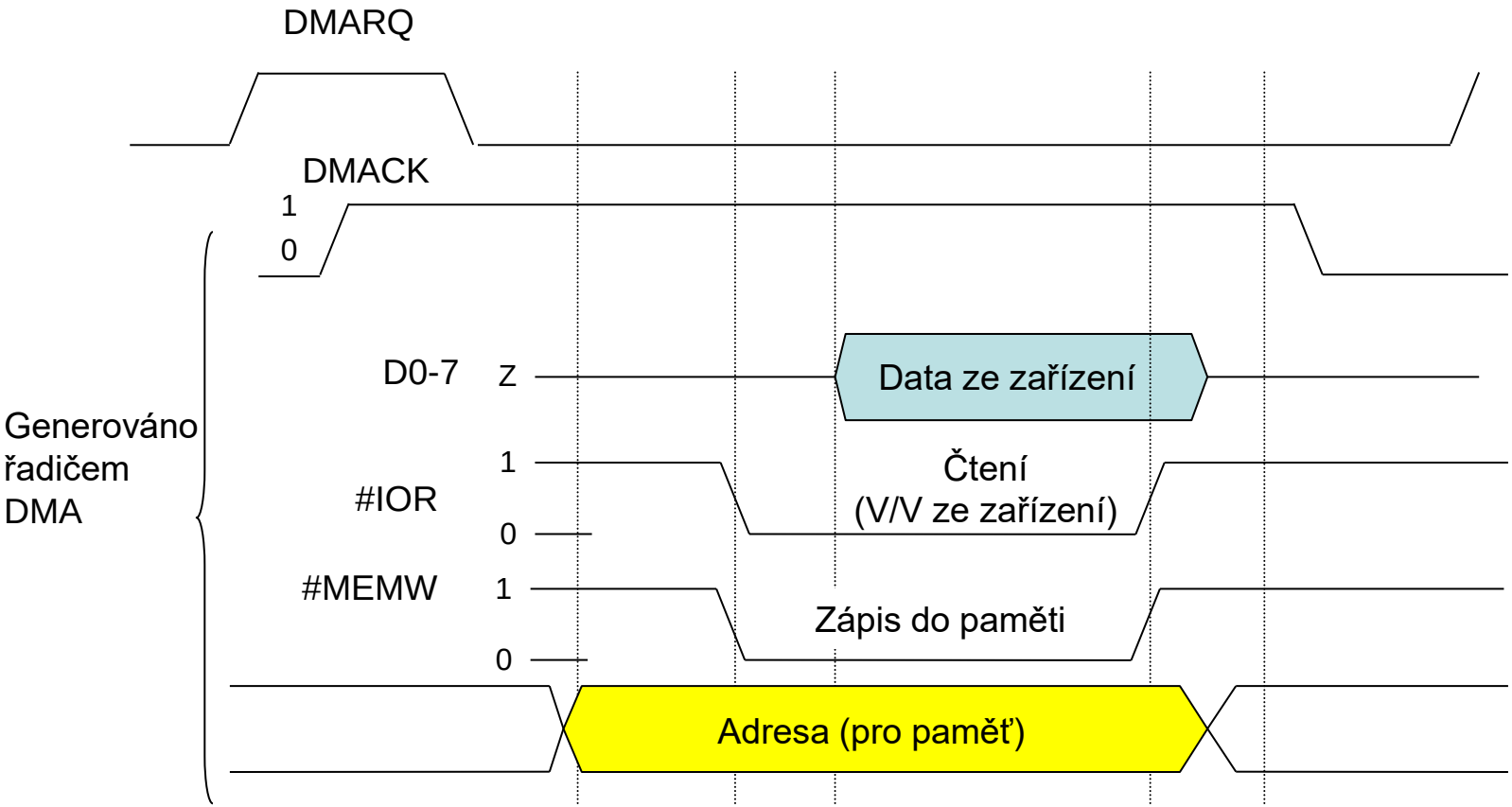
ATA časování





Časování vychází z časování sběrnice ISA, která byla systémovou sběrnicí počítačů PC AT

ATA DMA


DMARQ
DMACK
1
0
D0-7
Z
Data ze zařízení
Generováno řadičem DMA
1
#IOR
Čtení
(V/V ze zařízení)
0
#MEMW
1
Zápis do paměti
0
Adresa (pro paměť)

ATA – příkazový blok

Adresa	Čtení (byte)	Zápis (byte)
Báze+0	Data	Data
Báze+1	Error	Feature
Báze+2	Sector Count	Sector Count
Báze+3	LBA 0-7/24-31	LBA 0-7/24-31
Báze+4	LBA 8-15/32-39	LBA 8-15/32-39
Báze+5	LBA 16-23/40-47	LBA 16-23/40-47
Báze+6	Device/ LBA 24-27	Device/ LBA 24-27
Báze+7	Status	Command

Báze je typicky rovna 1F0h. V systémech s PCI sběrnicí lze bázi nutno vyčíst v BAR registru v konfigurační prostoru ATA řadiče.

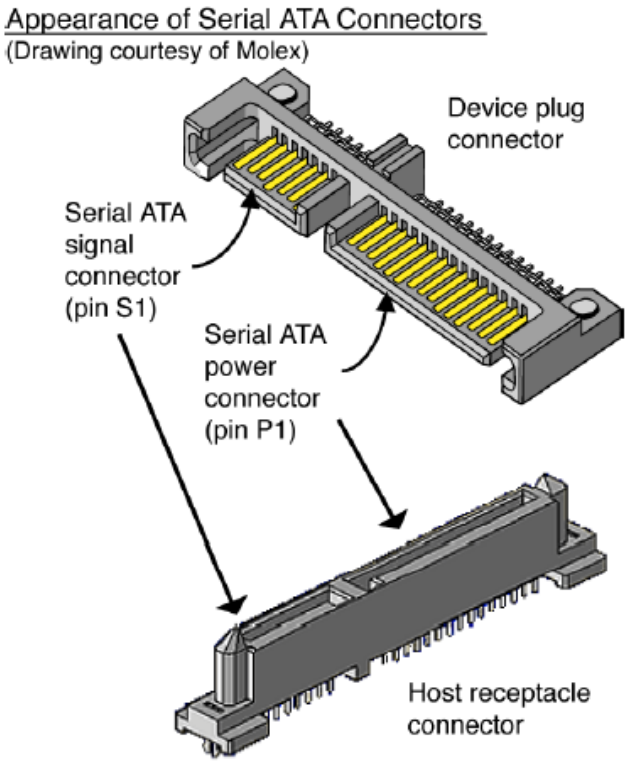
ATA příkazy

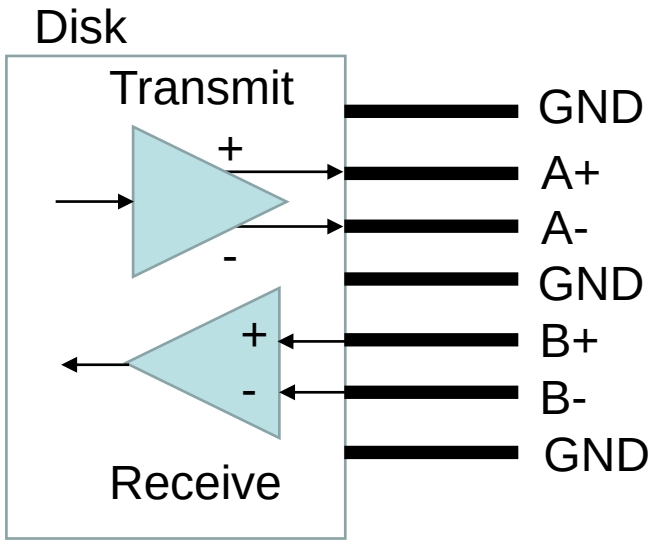
- READ SECTORS (20h)
  - Čtení sektorů v PIO módu
- READ DMA (C8h)
  - Čtení sektorů přes DMA
- WRITE SECTORS (30h)
  - Zápis sektorů
- WRITE DMA (CAh)
  - Zápis sektorů přes DMA

SATA III

- Nástupce ATA rozhraní
- Sériové rozhraní
  - 2x diferenciální linka, full duplex
  - Přenosová rychlost 1.5, 3, 6 Gb/s
  - Kódování 8/10bits
- ATA příkazy
- Podpora pro hot-plug
- Port Multipliers – možnost napojení více disků na jeden port
- Port Selectors – možnost přístupu více hostů do jednoho portu/disku

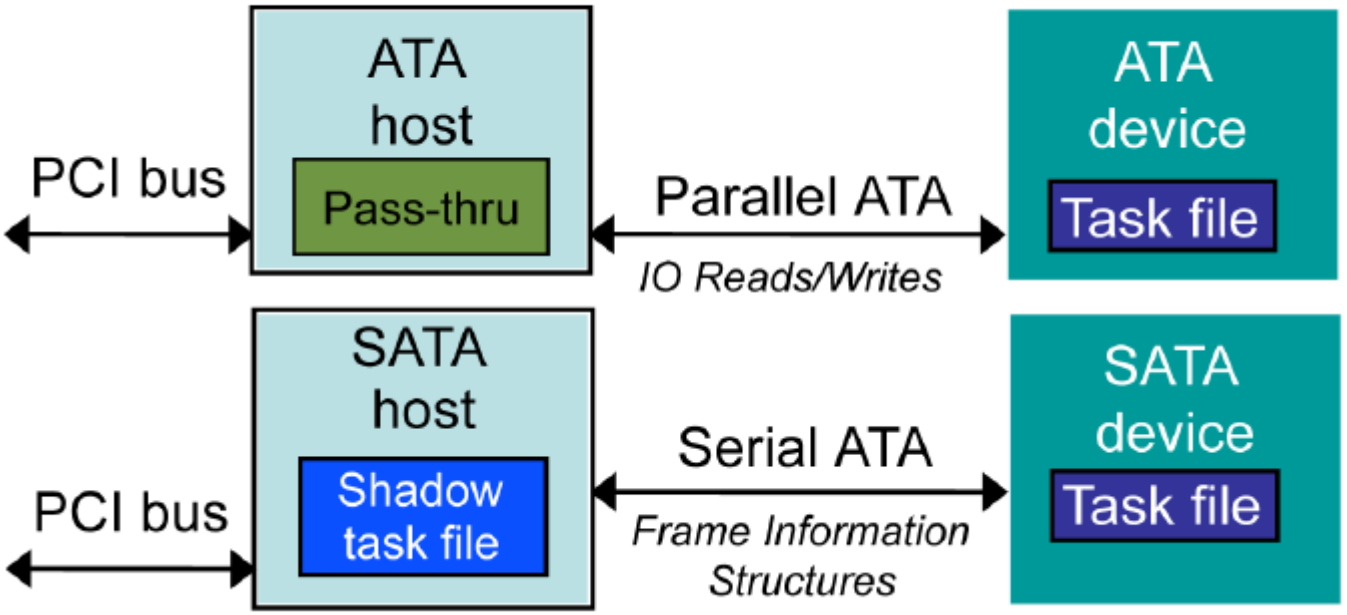
SATA konektor





Zdroj: SATA Storage Technology

ATA vs. SATA



Zdroj: SATA Storage Technology

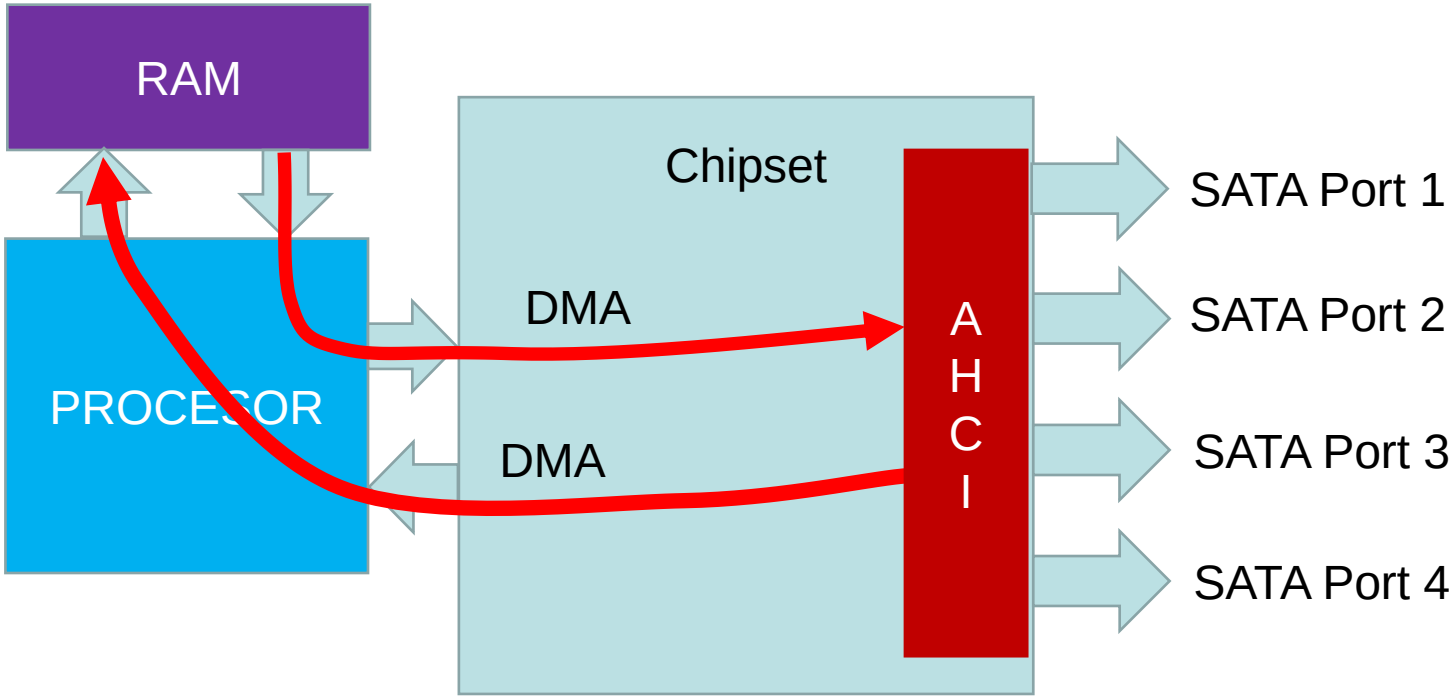
Frame Information Structure (FIS)

- Datové struktura, kterou se přenáší
  - ATA registry z řadiče do disku
  - ATA registry z disku do řadiče
  - Data
  - Další informace např. pro zajištění DMA přenosů
- FIS má různé formáty

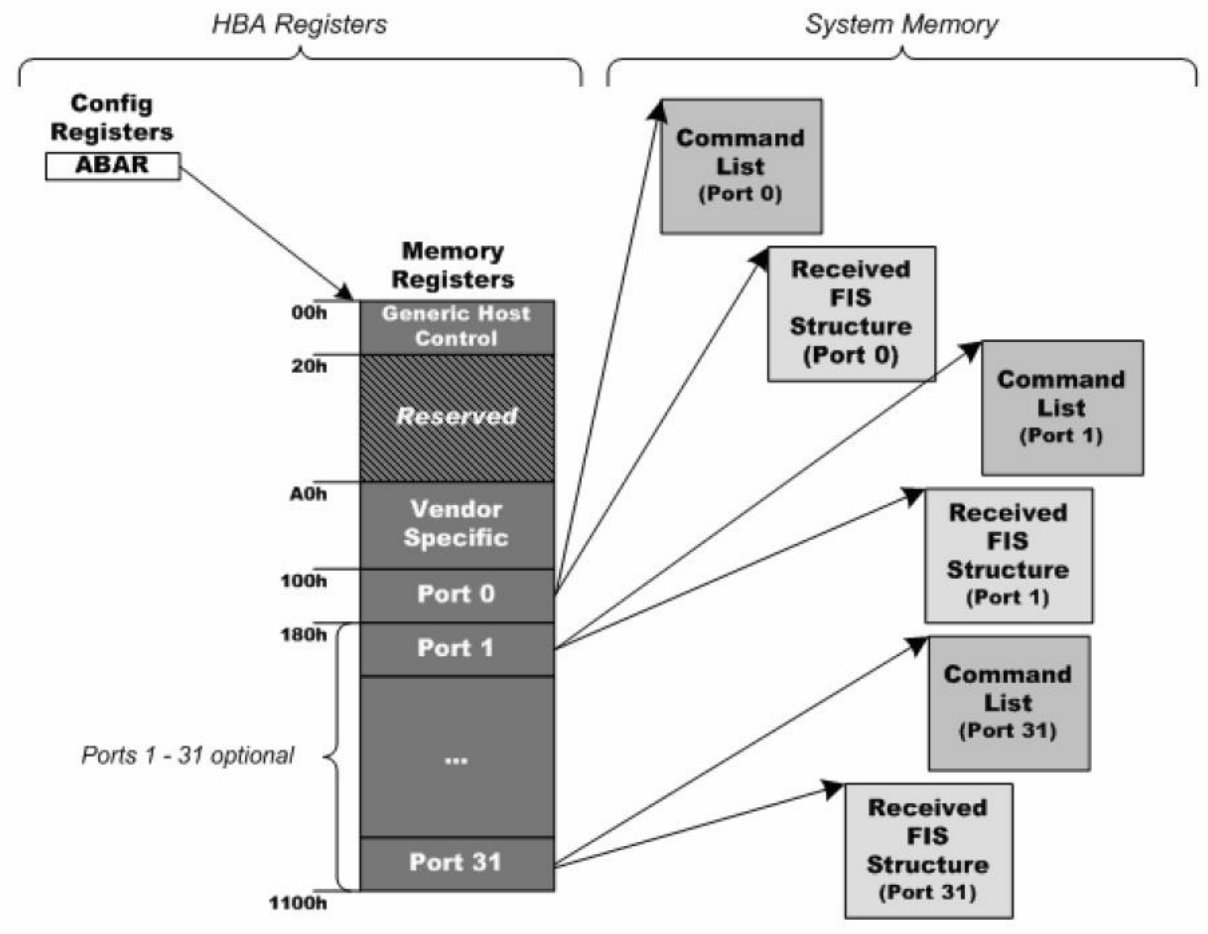
FIS pro přenos stínových registrů

Drive to host	Host to drive
ID(34h)	ID(27h)
Error	Feature
Sector Count	Sector Count
LBA 0-7/24-31	LBA 0-7/24-31
LBA 8-15/32-39	LBA 8-15/32-39
LBA 16-23/40-47	LBA 16-23/40-47
Device/ LBA 24-27	Device/ LBA 24-27
Status	Command

AHCI (Advanced Host Controller Interface)

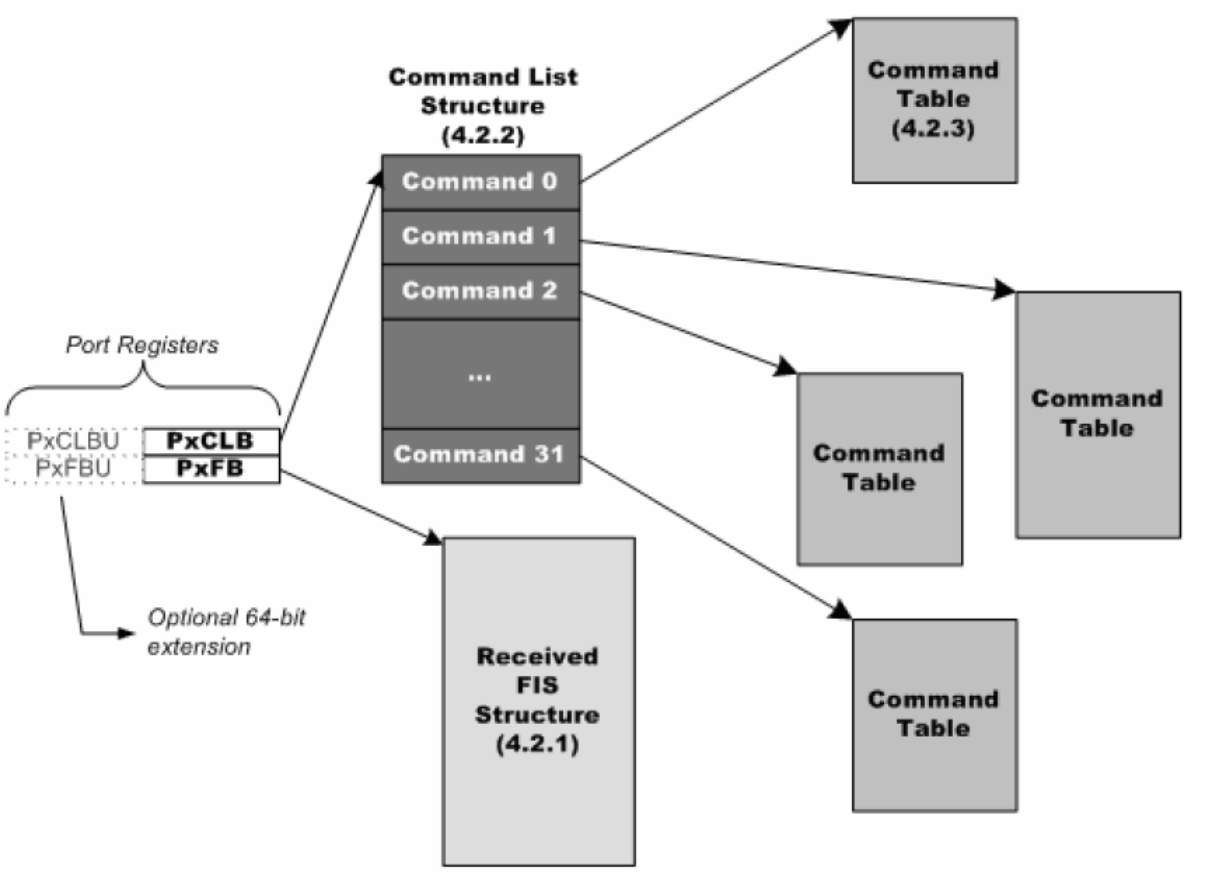


Hlavní paměťově mapovaná struktura



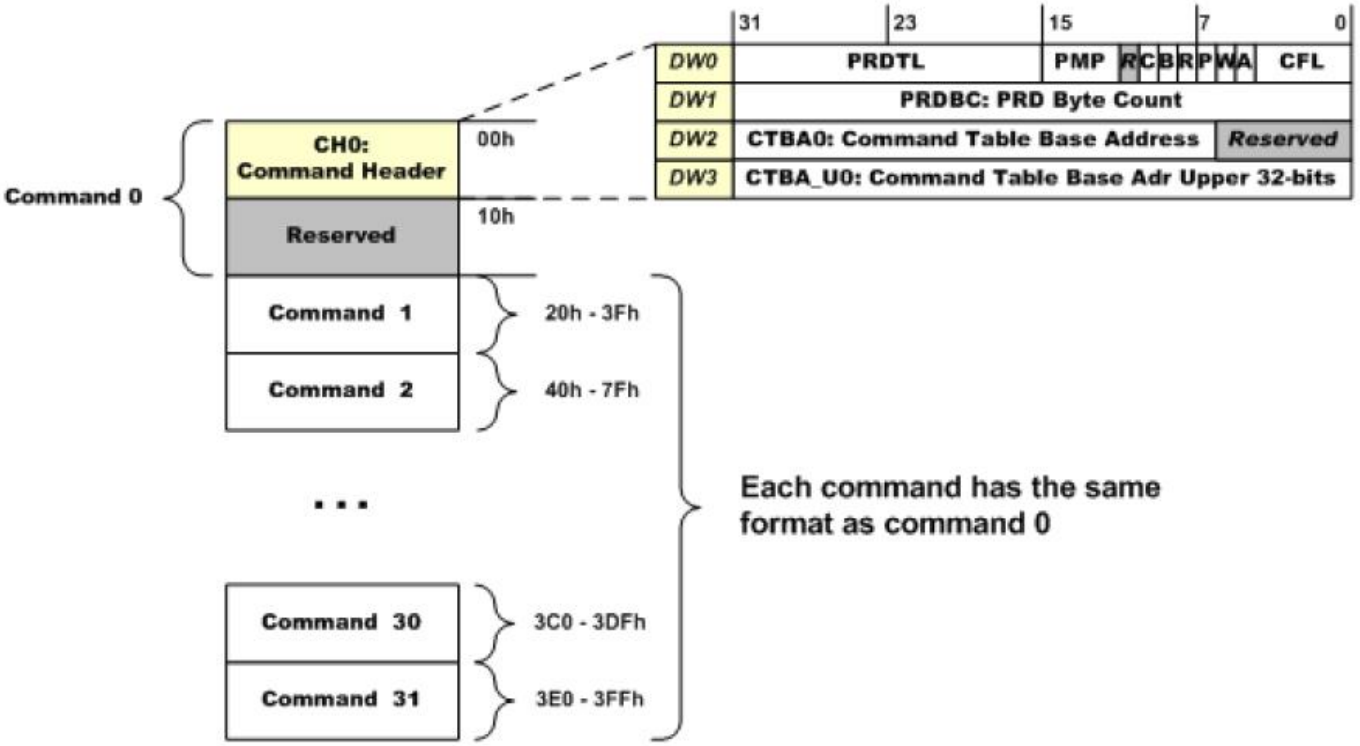
Zdroj: Serial ATA Advanced Host Controller Interface (AHCI) Revision 1.0

Seznam příkazů



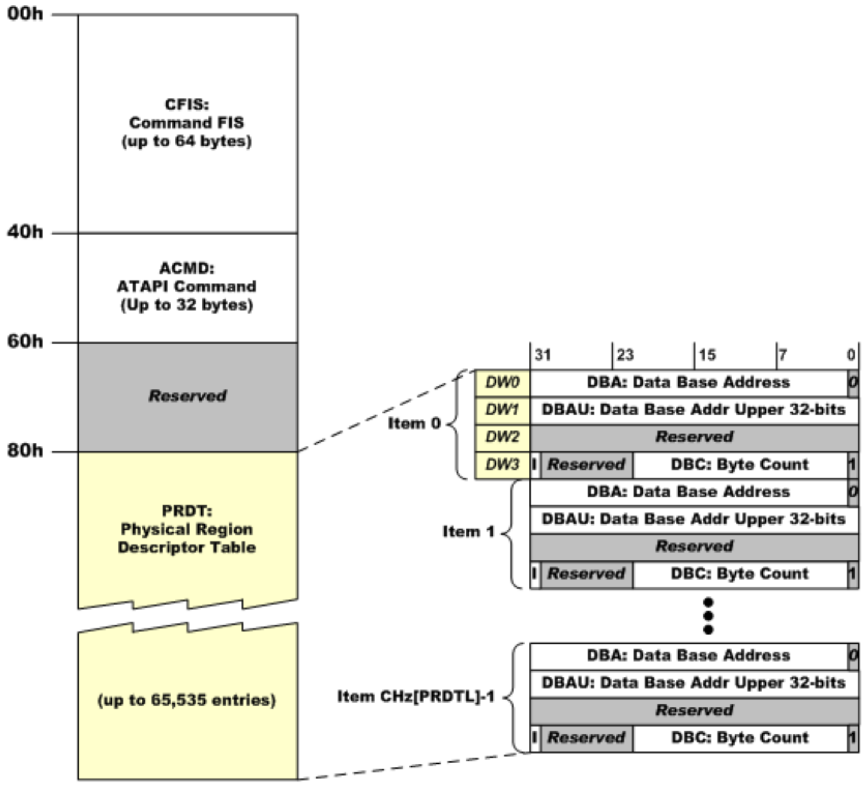
Zdroj: Serial ATA Advanced Host Controller Interface (AHCI) Revision 1.0

Příkazy



Zdroj: Serial ATA Advanced Host Controller Interface (AHCI) Revision 1.0

Tabulka příkazů



Zdroj: Serial ATA Advanced Host Controller Interface (AHCI) Revision 1.0

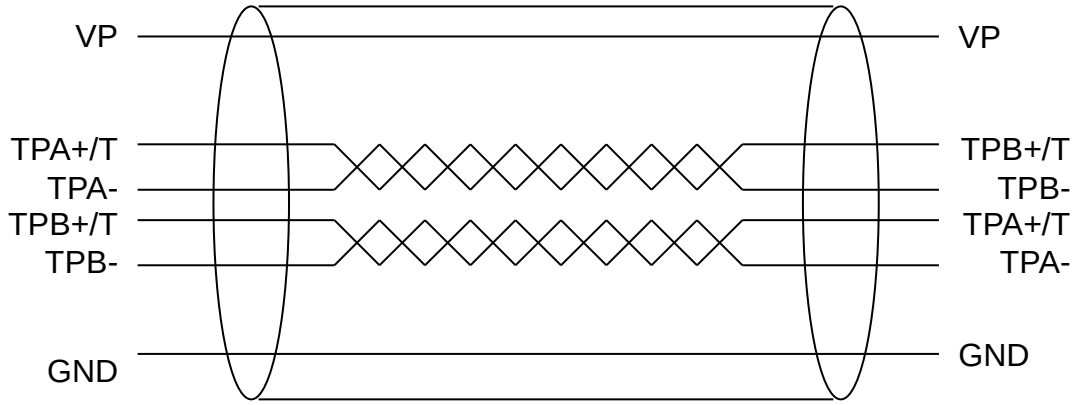
IEEE1394 FireWire

- Vysokorychlostní standardizovaná sběrnice
- Standard IEEE1394 přijat v roce 1995
- Rychlost přenosu 100 (S100), 200 (S200), 400 Mb/s (S400), 800 Mb/s S(800), 1600 Mb/s S(1600), 3200 Mb/s (S3200)
- Sběrnice typu peer-to-peer
- Asynchronní a izochronní přenosy dat
- Vhodná pro přenos audio i video dat

Aplikační oblasti

- Rozhraní počítačů Apple
- Rozhraní řady notebooků
- Připojení pro zařízení typu
  - DVD/CD, externí disky, CF čtečky
  - Fotoaparáty/kamery
  - Tiskárny/scannery
- Síť přes FireWire

Propojovací kabel FireWire



Kabel obsahuje dva kroucené překřížené páry pro přenos dat a dva napájecí vodiče (VP a GND). VP je v rozsahu 8-30V, 1,5A.

Typy přenosů

- Asynchronní
  - garantují doručení (při nedoručení se pokus o doručení opakuje)
- Izochronní
  - Garantují šířku pásma a zpoždění
  - Maximálně 64 izochronních kanálů
  - Zařízení naslouchají na všech kanálech a přebírají pouze data na požadovaných kanálech

Protokol IEEE1394

- Fyzická vrstva
  - Zajišťuje arbitráž
  - Zajišťuje serializaci a deserializaci dat
- Linková vrstva
  - Zajišťuje formování, vysílání a přijímání datových paketů
  - Zajišťuje potvrzování
- Transakční vrstva
  - Implementuje protokol request/response dle IEEE1212

IEEE1212

- FireWire implementuje transakční vrstvu dle IEEE1212
- IEEE1212 definuje architekturu paměťově mapovaných mikropočítačových sběrnic.
- Základní adresovatelnou entitou je uzel (node).
- Adresa uzlu má 64 bitů, z toho 16 bitů je identifikátor uzlu (ID node), 48 méně významných bitů je adresa do adresového prostoru uzlu. Adresní prostor uzlu obsahuje konfigurační ROM, kde jsou uloženy informace o uzlu ve standardizovaných datových strukturách.
- IEEE1212 definuje transakce
  - čtení/zápis (žádost)
  - Zamykací transakce
  - Status dokončení (potvrzení)

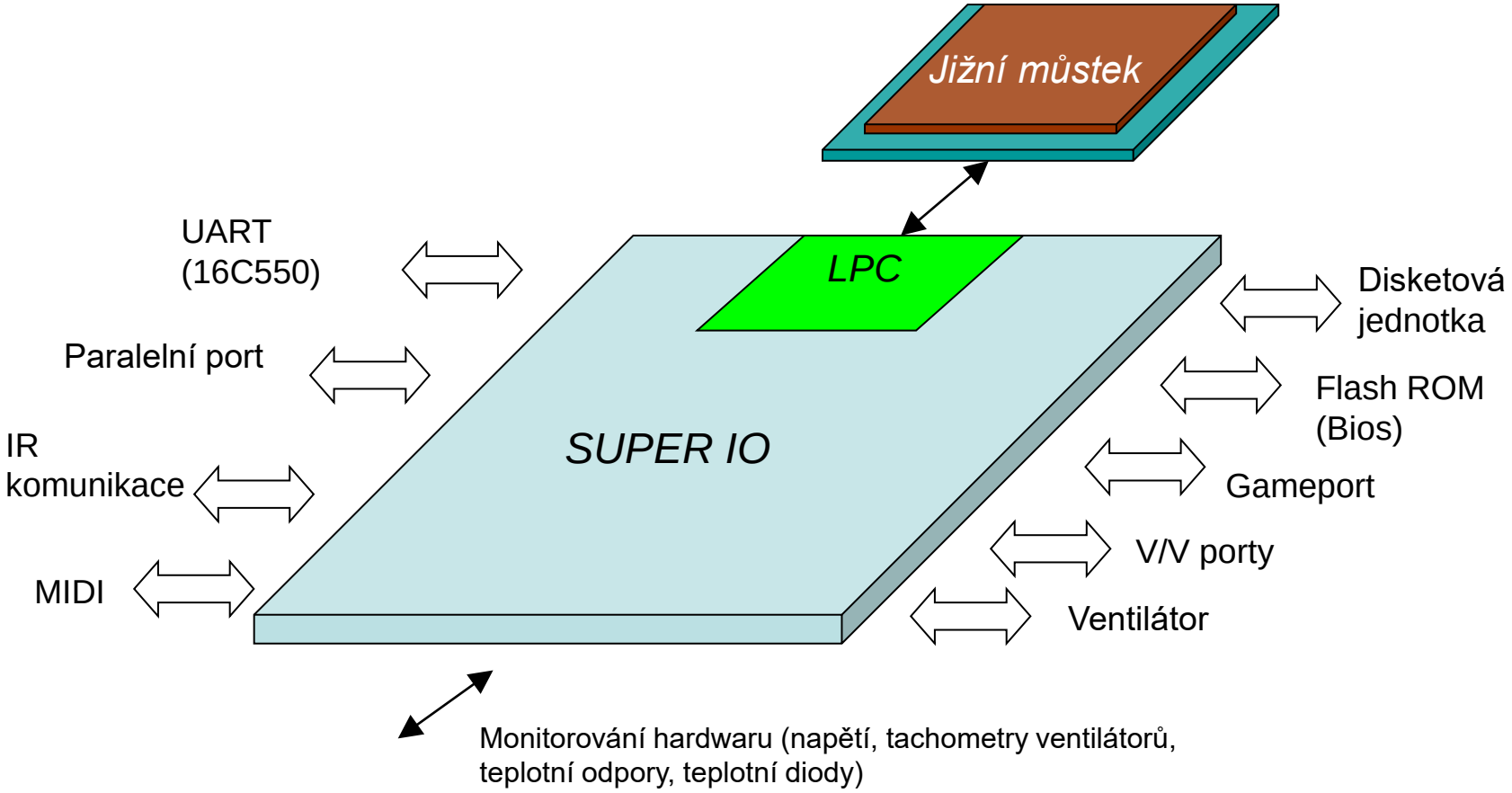
Protokol SBP-2

- SBP-2 (Serial Bus Protocol) slouží k transportu SCSI příkazů a dat přes rozhraní IEEE1394 (FireWire)
- Definuje obálku pro uložení SCSI příkazového bloku, stavového bloku a pro přenos dat (obdoba CBW a CSW u Mass Storage zařízení u USB)
- Je nadstavbou nad IEEE1212

Sběrnice LPC

- Sběrnice propojující CPU a zařízení nevyžadující vysokou datovou propustnost, a která byla dříve připojena přes sběrnici ISA
  - Boot ROM
  - Sériové porty
  - Řadič FD
  - Parallel port
- Typicky propojuje jižní můstek s tzv. SuperIO čipem (IT8705)
- Má pouze 7 signálů celkem, data jsou 4 bitová, adresa a data jsou multiplexovány
- Synchronní sběrnice

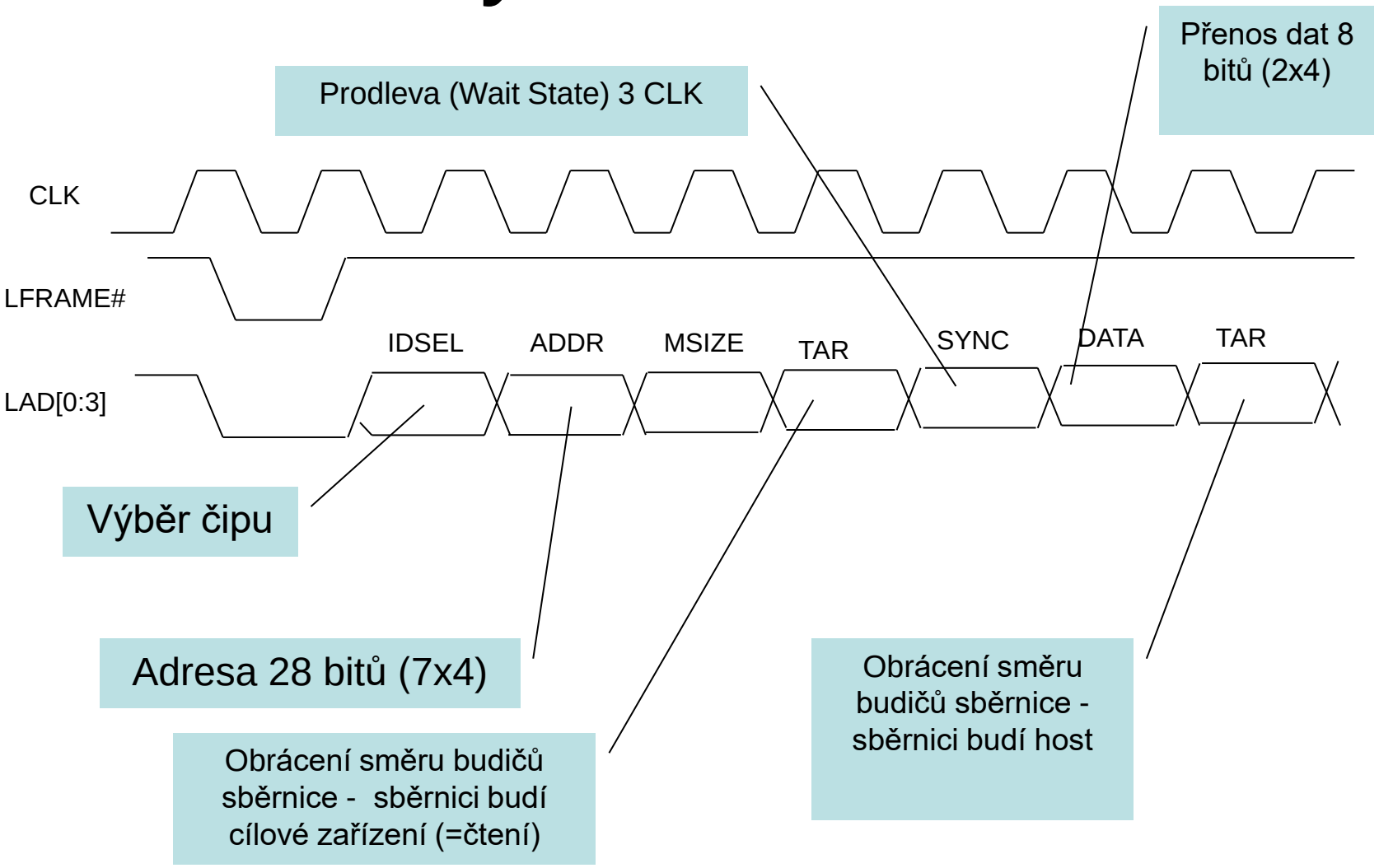
SuperIO


Jižní můstek
LPC
SUPER IO
UART
(16C550)
Paralelní port
IR
komunikace
MIDI
Disketová
jednotka
Flash ROM
(Bios)
Gameport
V/V porty
Ventilátor
Monitorování hardwaru (napětí, tachometry ventilátorů,
teplotní odpory, teplotní diody)

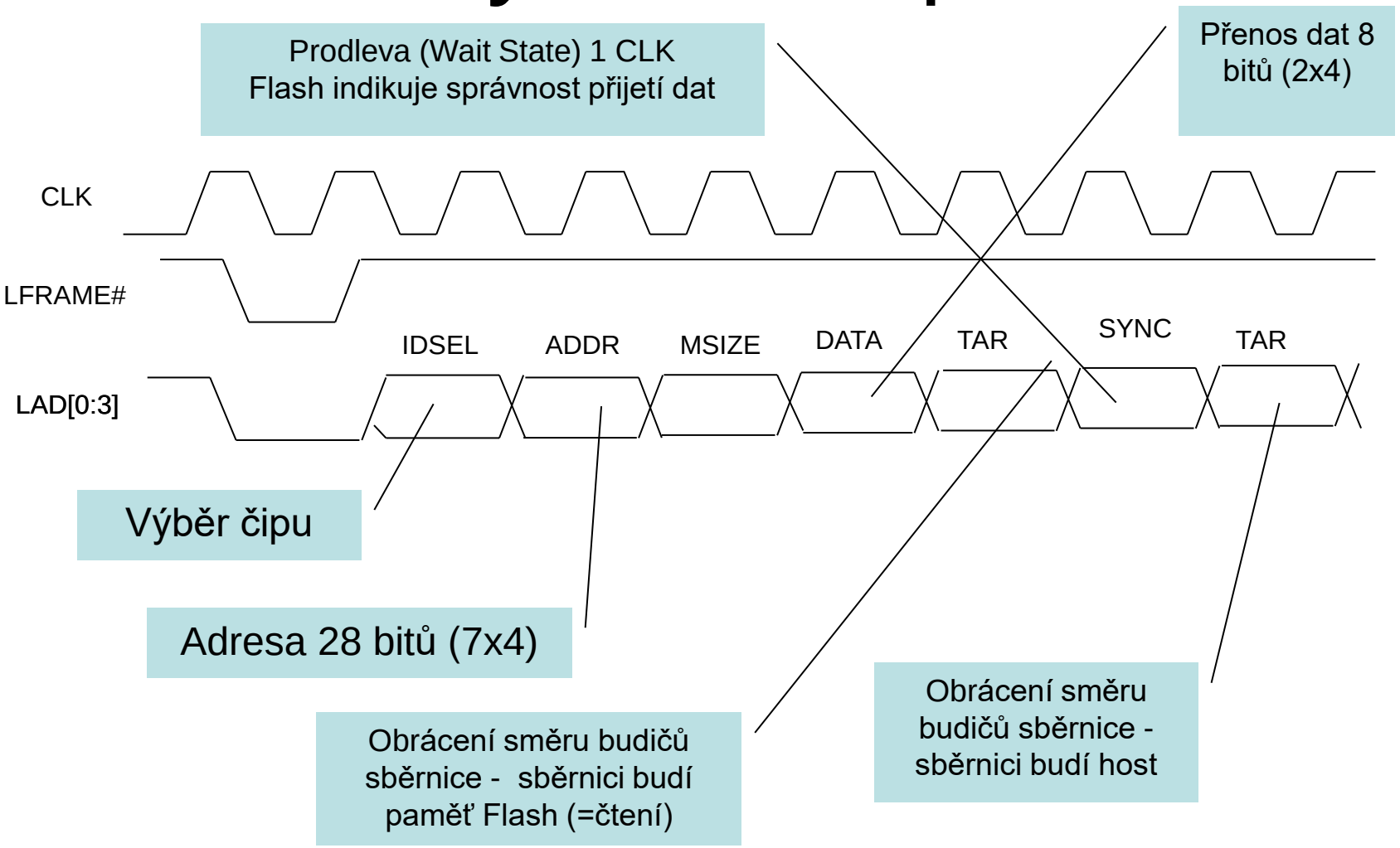
LPC signály

- LAD[0:3] – multiplexovaná adresa/data
- LFRAME# - začátek transakčního rámce
- LRESET# - reset signál
- LCLK – hodinový signál
- Nepovinné
  - LDRQ# - žádost o DMA nebo Bus Master požadavek
  - SERIRQ – řádost o přerušení
  - CLKRUN# - požadavek na spuštění/zastavení hodin (pro mobilní zařízení)
  - LPME# - probuzení z režimu nízké spotřeby

LPC cyklus – čtení Flash



LPC cyklus – zápis Flash



Metody připojování periferií

BI-MPP

Přednáška 7

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Realizace USB zařízení
- USB podsystém v PIC24FJ256GB106
- Příklady přenosů dat

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- Universal Serial Bus Specification 3.0, Revision 1.0, Listopad 2008 http://www.usb.org/developers/docs/usb_30_spec_092911.zip
- PIC24F Family Reference Manual, section 27, USB On-The-Go (OTG). Technická dokumentace Microchip Technology Incorporated. 2010 (http://www.microchip.com)
- PIC24FJ256GB110 Family Data Sheet. Technická dokumentace Microchip Technology Incorporated. 2009. (http://www.microchip.com)

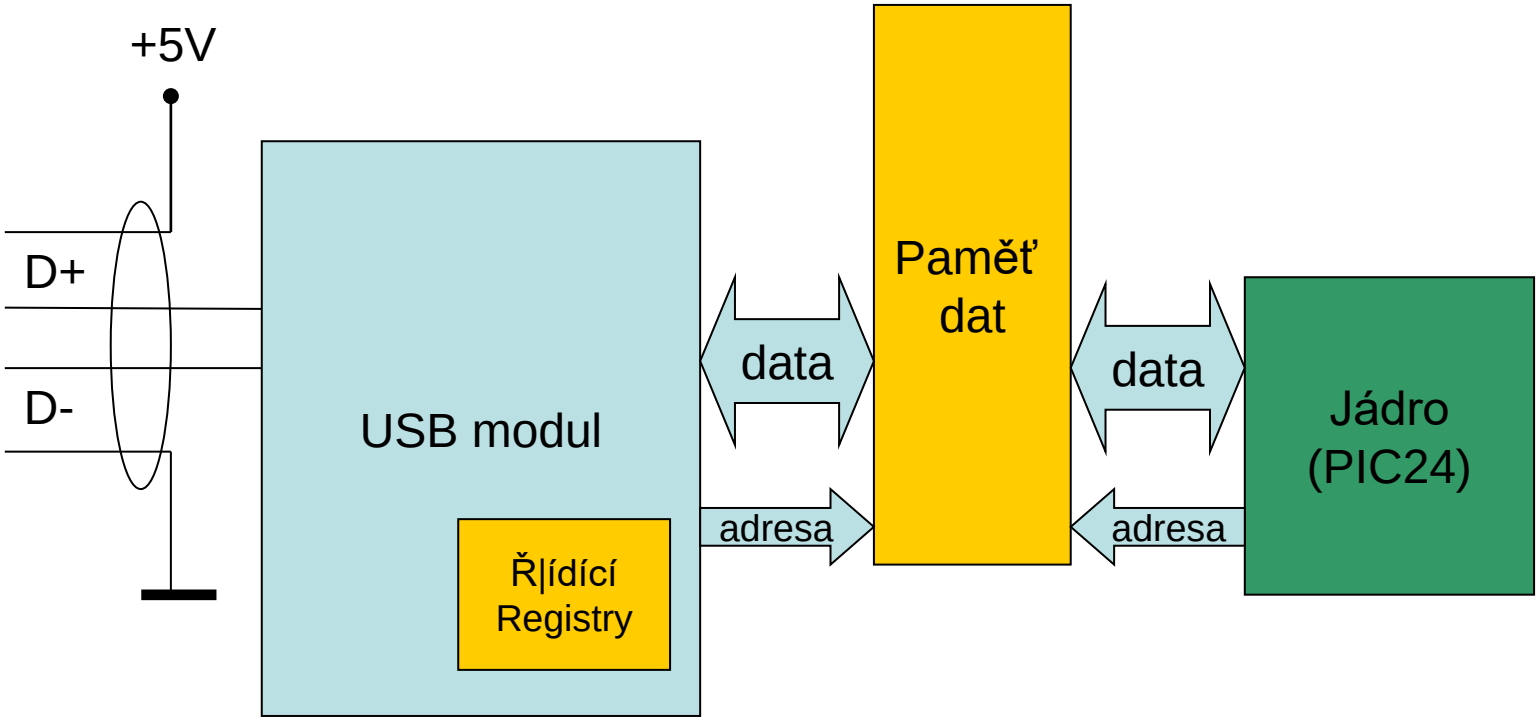
Realizace USB zařízení

- USB řadiče a mikrokontroléry podporují
  - USB zařízení
  - USB host (On-the-Go)
- USB rozhraní a jeho programová obsluha závisí na konkrétním výrobci řadiče nebo mikrokontroléru
- Výrobci poskytují vzorové programové vybavení pro USB, které návrháři upravují pro potřeby vyvíjeného zařízení

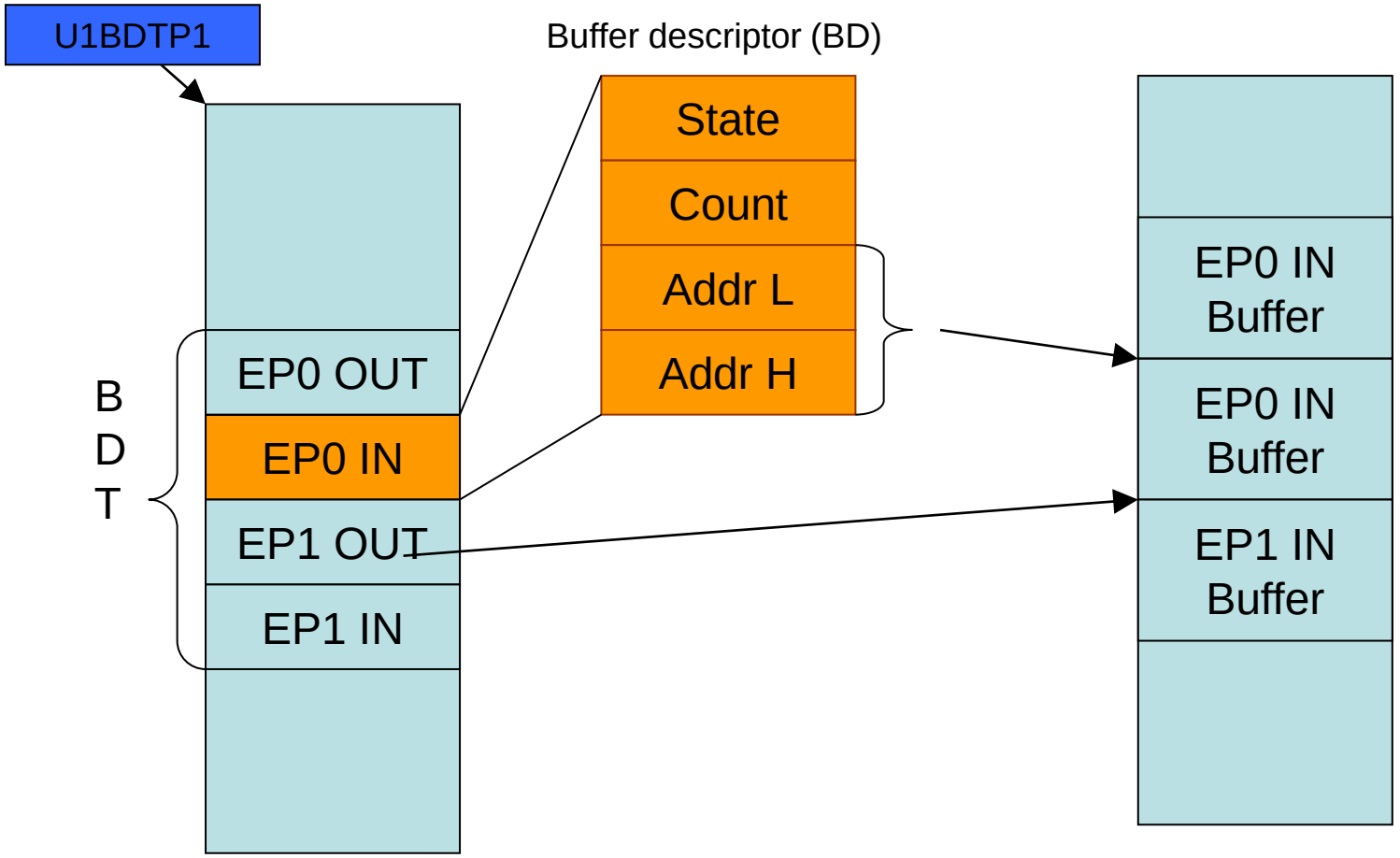
USB v PIC24FJ256GB106

- Hardware zajišťuje serializaci a deserializaci dat, detekci a řízení stavu linky (D+, D-)
- Data jsou ukládána/čtena do/z bufferů, které jsou definovány pro každý koncový bod zvlášť
- Dokončení přenosů dat, příjem speciálních paketů (např. SOF) a změny stavu linky vyvolávají přerušení
- Konfigurace USB řadiče se provádí přes registry řadiče

USB modul v PIC24FJ256GB106



Popisovače v USB modulu

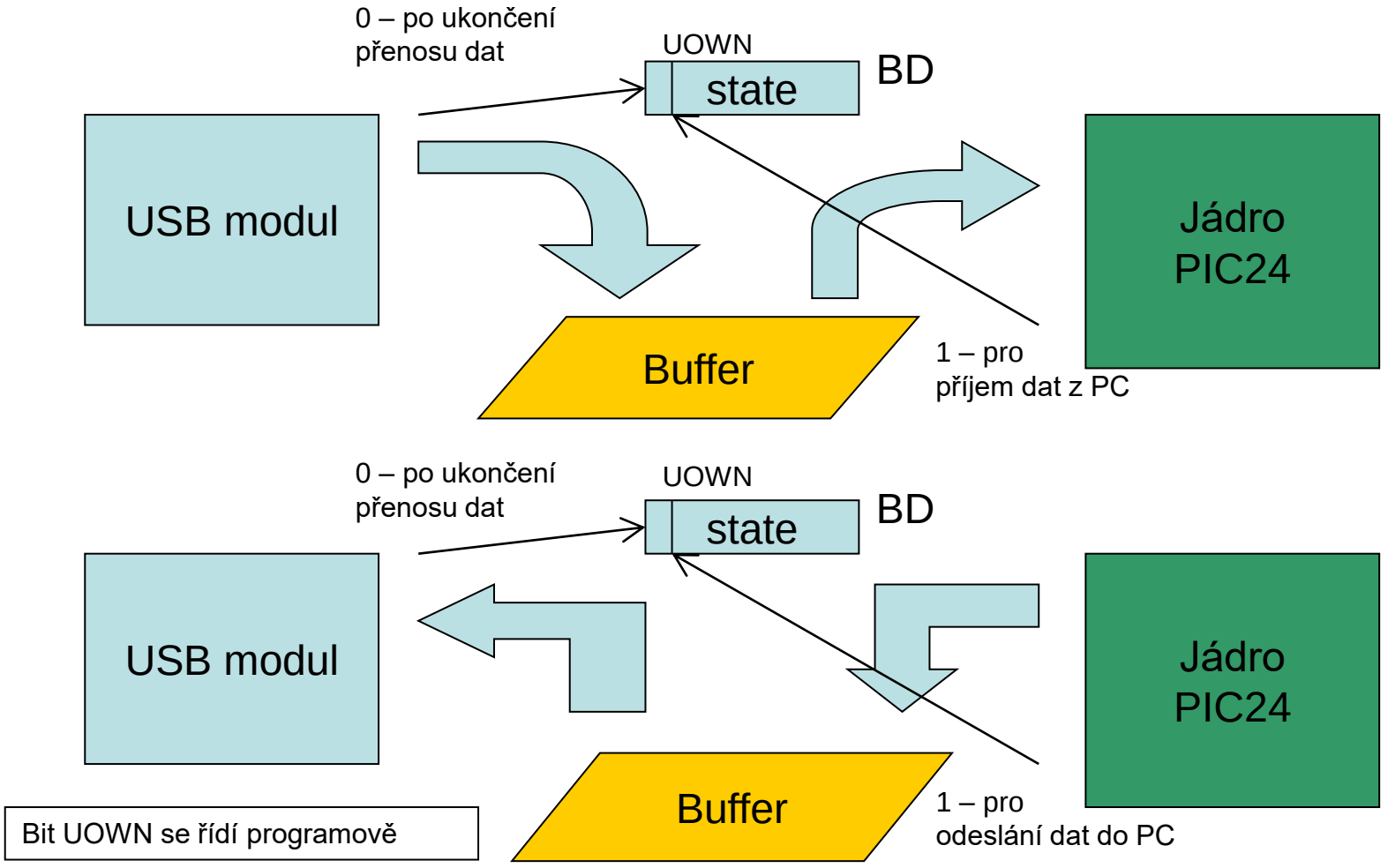


BDT – buffer descriptor table (tabulka popisovačů bufferů)
BD – buffer descriptor (popisovač bufferu)

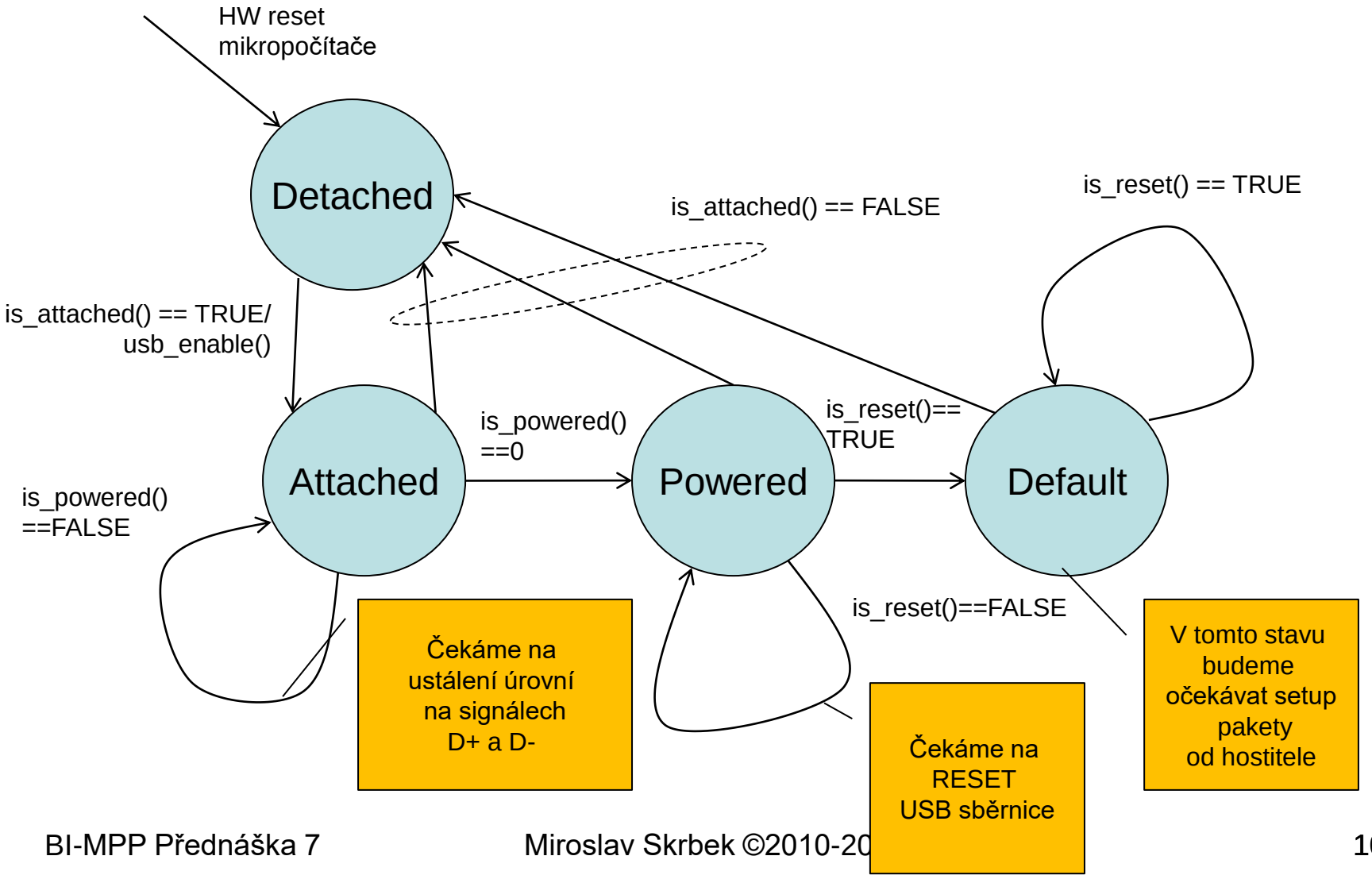
BDT (Buffer Descriptor Table)

- Tabulka obsahuje pro každý koncový bod a směr popisovač bufferu
- Popisovač bufferu obsahuje
  - Stav bufferu (state)
  - Počet přijatých/k odeslání bytů (count)
  - Adresa bufferu v paměti (Addr L, Addr H)
- Bit 7 stavu bufferu se synchronozuje přístup do bufferu ze strany USB řadiče a CPU

Řízení přístupu do bufferu



Stavy zařízení (zjednodušeno)



Další stavy zařízení

- Addressed
  - Do tohoto stavu se přechází ze stavu Default, když zařízení obdrží žádost o nastavení adresy zařízení a nastaví přijatou adresu v USB subsystému (usb_set_address(…))
- Configured
  - Do tohoto stavu zařízení přechází ze stavu addressed po přijetí žádosti SET_CONFIGURATION

Knihovna USBLIB

- Knihovna základních funkcí pro ovládání USB modulu v mikropočítačích PIC24
- Poskytuje funkce pro
  - Vytvoření tabulky popisovačů bufferů (BDT)
  - Funkce pro detekci stavu USB modulu
  - Funkce pro zahájení přenosu dat z/do PC
  - Funkce pro kopírování dat z/do bufferů
- Knihovna byla vyvinuta pro potřeby výuky na FIT a FEL ČVUT v Praze, včetně portování pro mikropočítače řady PIC24F

Deklarace koncových bodů a vytvoření BDT

Deklarace BDT

```
DECLARE_EP_BEGIN
        DECLARE_EP(ep0);    // IN/OUT Control EP
        DECLARE_EP(ep1);    // IN Interrupt EP
        DECLARE_EP(ep2);    // OUT data EP
        DECLARE_EP(ep3);    // IN data EP
DECLARE_EP_END
```

Deklarace bufferů

```
DECLARE_BUFFER_BEGIN
        DECLARE_BUFFER(ep0_buf_out, EP0_OUT_BUF_SIZE);
        DECLARE_BUFFER(ep0_buf_in, EP0_IN_BUF_SIZE);
        DECLARE_BUFFER(ep1_buf_in, EP1_IN_BUF_SIZE);
        DECLARE_BUFFER(ep2_buf_out, EP2_OUT_BUF_SIZE);
        DECLARE_BUFFER(ep3_buf_in, EP3_IN_BUF_SIZE);
DECLARE_BUFFER_END
```

Funkce pro zjištění stavu USB modulu

- `int is_attached();`
  vrací 1, pokud je zařízení připojeno k USB
- `int is_powered();`
  vrací 1, pokud je USB datová linka ve stavu SE0
- `int is_reset();`
  vrací 1, pokud byl detekován reset na USB sběrnici. Volání funkce příznak nuluje.
- `int is_idle();`
  detekuje požadavek přechodu do stavu snížené spotřeby
- `int is_sof();`
  detekuje příchod SOF paketu
- `int is_transfer_done();`
  vrací 1, pokud byl dokončen přenos. Volání funkce příznak nenuluje.
- `int get_trn_status();`
  vrací číslo koncového bodu, kde došlo k dokonšení přenosu. Tato funkce nuluje příznak dokončeného přenosu.

Funkce pro práci s endpointy a buffery

- **`extern void usb_init_ep(int num, byte ep_type, ep_buf_dsc_t* ep);`**
  inicializuje koncový bod
- **`void usb_ep_transf_start(volatile ep_buf_dsc_t* ep, byte transf_type, buf_ptr_t buffer, int size);`**
  zahajuje přenos dat z/do PC
- **`void copy_to_buffer(buf_ptr_t buf, byte* src, int size);`**
  kopíruje data z paměti do bufferu
- **`void copy_from_buffer(buf_ptr_t buf, byte* dest, int size);`**
  kopíruje data z bufferu do paměti

Přenos do PC - příklad

Následující příklad ukazuje přenos odezvy na žádost GET_DESCRIPTOR – Device Descriptor do zařízení

```
// Inicializace koncového bodu
usb_init_ep(0, EP_SETUP_INOUT, &__bdt.ep0);

copy_to_buffer(ep0_buf_in, &device_descriptor,
   sizeof(device_descriptor));

// Zahájení přenosu dat do PC (očekává se SETUP paket)
usb_ep_transf_start(EP(ep0), USB_TRN_DATA1_IN, ep0_buf_in,
                sizeof(device_descriptor));

// čekej dokud není přenos dokončen
while (!is_transfer_done());

// zjištění koncového bodu, kde došlo k dokončení přenosu
int epnum = get_trn_status();
if (epnum == 0x00) {
        // naplánuj příjem potvrzení od PC
}
```

Přenos z PC - příklad

Následující příklad ukazuje přenos požadavku (USB DEVICE REQUEST) do zařízení

```
// Inicializace koncového bodu
usb_init_ep(0, EP_SETUP_INOUT, &__bdt.ep0);

// Zahájení přenosu dat z PC (očekává se SETUP packet)
usb_ep_transf_start(&__bdt.ep0, USB_TRN_SETUP, ep0_buf_out,
        EP0_OUT_BUF_SIZE);

// čekej dokud není přenos dokončen
while (!is_transfer_done());

// zjištění koncového bodu, kde došlo k dokonční přenosu
int epnum = get_trn_status();
if (epnum == 0x00) {
        // data přijata na EP0, zkopírujeme obsah bufferu
        // do proměnné req (usb device request)
        copy_from_buffer(ep0_buf_out, (byte*)req,
                sizeof(usb_device_req_t));
}
```

Metody připojování periferií

BI-MPP

Přednáška 8

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Jádro operačního systému Linux
- Moduly jádra
- Ovladač znakového zařízení

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- Corbet, J., Rubini A., Kroah-Hartman, G.: Linux Device Drivers. Third Edition. O'Reilly Media 2005. ISBN: 0-596-00590-3.

Jádro operačního systému Linux

- Napsané v jazyce C, pouze několik málo specifických částí je napsáno v assembleru
- Jádro je volně ke stažení ve zdrojové formě a je šířeno pod GNU licencí
- Podporuje velké množství jak procesorů (x86, PPC, ARM, …), tak platforem (PC, PDA, …)
- Obsahuje ovladače pro velké množství zařízení
- Překládá se překladačem GCC a umožňuje křížovou kompilaci (např. na PC můžeme přeložit jádro pro procesor ARM)

Konfigurace jádra

- Jádro Linuxu vyžaduje před překladem konfiguraci, která vybírá
  - Procesor
  - Platformu
  - Ovladače
- U ovladačů je možno v konfiguraci zvolit
  - nezařadit (nekompilovat)
  - zakompilovat do jádra (ovladače nezbytné pro start OS, např. ovladače disků, podpora file systémů)
  - Přeložit jako moduly jádra (možnost dynamického zavádění)

Moduly jádra Linuxu

- Ovladače jsou moduly do jádra operačního systému Linux
- Moduly jsou psané v jazyce C (kritické části mohou být i v assembleru, ale je to spíše výjimečné)
- Modul je překládán až do úrovně object souboru (*.o) a proto berou moduly jádra příponu *.ko.
- Modul ve formě object souboru je přilinkován za běhu do jádra (pozor ! *.ko není totéž jako *.so, protože *.so je produktem linkeru (ld))

Příklad modulu

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE(“Dual BSD/GPL“)

static int mo_init(void) {
        printk(KERN_ALERT “inicializace\n”);
}

static int mo_exit(void) {
        printk(KERN_ALERT “ukonceni\n”);
}

module_init(mo_init)
module_exit(mo_exit)
```

Komentář k příkladu modulu

- Makro module_init registruje funkci mo_init jako inicializační funkci. Tato funkce je volána jen jednou, když je zaveden modul do jádra příklazem insmod nebo modprobe.
- Makro module_exit registruje funkci mo_exit jako deinicializační funkci. Tato funkce je volána jen jednou, když je modul odstraněn z jádra příklazem rmmod.
- Funkce printk vytiskne zprávu, kterou je možno zobrazit pomocí příkazu dmesg. Velmi užitečné pro ladění driverů.
- Makro MODULE_LICENCE registruje název použité licence. Text se stává součástí kódu modulu.

Překlad modulu mimo jádro a zavedení modulu do jádra

Makefile

Jméno(a) modulu(ů) s příponou *.o, musí se shodovat se jménem zdrojového souboru (mpp_module.c)

```
obj-m := mpp_module.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)
default:
	$(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
```

Překlad modulu

```
make
```

Zavedení modulu

```
insmod mpp_module.ko
```

Registrace znakového zařízení

```
struct file_operations
mydev_fops = {
.owner = THIS_MODULE,
.llseek = mydev_llseek,
.read = mydev_read,
.write = mydev_write,
.ioctl = mydev_ioctl,
.open = mydev_open,
.release = mydev_release,
};
```

Struktura file_operations obsahuje ukazatele na callback funkce, které jsou implementacemi souborových operací na straně driveru

Registrace znakového zařízení (v probe)

```
int register_chrdev(unsigned int major, const char *name,
struct file_operations *fops);
```

Funkce souborových operací I

Otevření souboru (open)
[vrací 0 po uspěšně provedené operaci, jinak chybový kód]

```
static int mpp_open(struct inode *inode, struct file *filp)
{ ... return 0; }
```

Uzavření souboru (close)
[vrací 0 po uspěšně provedené operaci, jinak chybový kód]

```
static int mpp_release(struct inode *inode, struct file *f)
{ ... return 0; }
```

Funkce souborových operací II

Čtení souboru (read)
[vrací počet přečtených bytů, tj. bytů zapsaných do bufferu. V ideálním případě by funkce měla vrátit size, tj. počet bytů požadovaných aplikací. Není to ale podmínkou.]

```
static int mpp_read(struct file *f, char *buf,
   size_t size, loff_t *offset)
{ ... return size; }
```

Zápis do souboru (write)
[vrací počet zapsaných bytů, tj. bytů přečtených z bufferu. V ideálním případě by funkce měla vrátit size, tj. počet bytů požadovaných aplikací k zapsání. Není to ale podmínkou.]

```
static int mpp_write(struct file *f, const char *buf,
       size_t size, loff_t *offset)
{ ... return size; }
```

User Space a Kernel Space

- Aplikace běží v uživatelském prostoru (UserSpace), který má řadu omezení (ochrany paměti např.)
- Jádro a tedy i ovladače běží v prostoru jádra, který má všechna privilegia
- V prostoru jádra běží program supervizorském režimu a aplikace běží v uživatelském režimu
- Při volání jádra (system call, např. kvůli souborové operaci) dochází k přepnutí z User Space na Kernel Space a při návratu zase opačně
- Uvědomte si, že User Space a Kernel Space reprezentují odlišné paměťové prostory, tudíž ukazatele (pointry) v jednom prostoru neplatí v druhém a naopak.

Kopírování dat
kernel space ↔ userspace

Kernel space → user space

```
unsigned long copy_to_user(void __user *to,
const void *from,
unsigned long count);
```

User space → kernel space

```
unsigned long copy_from_user(void *to,
const void __user *from,
unsigned long count);
```

Nejčastěji se používají v callback funkcích pro souborové operace read a write

Alokace dynamické paměti v jádře

Alokace paměťového bloku

```
char* kbuf = kmalloc(size, GFP_KERNEL);
```

Typ bloku paměti

Velikost

Uvolnění paměťového bloku

```
kfree(kbuf);
```

Komunikace s ovladačem ze strany aplikace (bash)

Zápis

```
echo -n "ahoj" >/dev/mpp
```

Čtení

```
cat /dev/mpp
```

Komunikace s ovladačem ze strany aplikace (programově - C)

Prototypy funkcí

```
#include <unistd.h>

int open(const char *pathname, int flags);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
int close(int fd);
```

Příklad

```
#include <unistd.h>

int main() {
  char buf[100];
  int fd = open("/dev/mpp", O_RDWR);
  read(fd, buf, 10);
  write(fd, buf, 10);
  close(fd);
}
```

Metody připojování periferií

BI-MPP

Přednáška 9

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Ovladače USB zařízení v Linuxu
- Windows Driver Model

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- Corbet, J., Rubini A., Kroah-Hartman, G.: Linux Device Drivers. Third Edition. O'Reilly Media 2005. ISBN: 0-596-00590-3.
- WDK Documentation, Windows Drivers Kits, Microsoft Corporation, 2009.

URB

- USB request block
- Datová struktura reprezentující USB přenosy
- Používá se pro posílání nebo přijímání dat z daného endpointu
- Zajišťuje asynchronní přenosy
- Obsahuje
  - Číslo endpointu
  - Délku přenosu
  - Adresu bufferu (pro příjem nebo odeslání dat)
  - Stav přenosu
  - Adresu dokončovací rutiny, která se volá po ukončení přenosu
- Vytváří se a ruší
  - struct urb *usb_alloc_urb(int iso_packets, int mem_flags);
  - void usb_free_urb(struct urb *urb);

Příprava URB

- Pro vyplnění URB slouží funkce
  - Interrupt transfers

```
void usb_fill_int_urb(struct urb *urb, struct
usb_device *dev,unsigned int pipe, void*
transfer_buffer, int buffer_length, usb_complete_t
complete, void *context, int interval);
```

  - Bulk transfers

```
void usb_fill_bulk_urb(struct urb *urb, struct
usb_device *dev, unsigned int pipe, void*
transfer_buffer, int buffer_length, usb_complete_t
complete, void *context);
```

  - Control transfers

```
void usb_fill_control_urb(struct urb *urb, struct
usb_device *dev, unsigned int pipe, unsigned char
*setup_packet, void *transfer_buffer, int
buffer_length
```

Odeslání URB ke zpracování

- **`int usb_submit_urb(struct urb *urb, int mem_flags);`**
  - Parametr urb je ukazatel na URB
  - Parametr mem_flags se nastavuje podle kontextu, ve kterém funkce je použita
    - GP_ATOMIC
    - GP_NOIO
    - GP_KERNEL
- Zpracování URB lze zastavit funkcemi
  - int usb_kill_urb(struct urb *urb);
  - int usb_unlink_urb(struct urb *urb); (nečeká na dokončení)

Informace o podporovaných zařízeních driverem

```
static struct usb_device_id mydev_table [ ] = {
{ USB_DEVICE(USB_MYDEV_VENDOR_ID, USB_MYDEV_PRODUCT_ID) },
{ } /* ukončující položka */
};
MODULE_DEVICE_TABLE (usb, mydev_table);
```

Pokud driver podporuje více zařízení s různými VID a PID, pak má tabulka více položek. Tabulka se používá pro vyhledání driveru pro dané zařízení.

Registrace USB driveru

```
static struct usb_driver my_driver = {
.owner = THIS_MODULE,
.name = „myusbdev",
.id_table = mydev_table,
.probe = mydev_probe,
.disconnect = mydev_disconnect,
};
```

Registrace driveru uvnitř inicializační funkce modulu

```
result = usb_register(&mydev_driver);
if (result)
   err("usb_register failed. Error number %d",
result);
```

Probe

Funkce probe slouží k ověření zařízení z hlediska schopnosti driveru ovládat toto zařízení

Vytváří vnitřní lokální struktury pro konkrétní (instanci) zařízení a registruje ho v jádře

Vrací 1 pokud je zařízení úspěšně registrováno

Registrace USB zařízení v Probe

```
retval = usb_register_dev(interface, &mydev_class);
if (retval) {
        err("Not able to get a minor for this device.");
        usb_set_intfdata(interface, NULL);
        goto error;
}
```

Přenosy bez URB

```
int usb_bulk_msg(struct usb_device *usb_dev, unsigned int
 pipe,void *data, int len, int *actual_length,
 int  timeout);

int usb_control_msg(struct usb_device *dev,
  unsigned int pipe,__u8 request, __u8 requesttype,
  __u16 value, __u16 index, void *data, __u16 size,
  int timeout);
```

Zjednodušují psaní driveru, pro realizaci přenosu stačí jedno volání funkce

usb_bulk_msg je pro blokové přenosy a usb_control_msg je pro řídící přenosy

Windows Driver Model

- Model ovladačů pro OS Windows, který podporuje Plug&Play zařízení
- K dispozici je od Windows 98
- Vývoj ovladačů probíhá v jazyce C (včetně knihovní podpory)
- Pro vývoj ovladačů je k dispozici Windows Driver Kit (WDK)

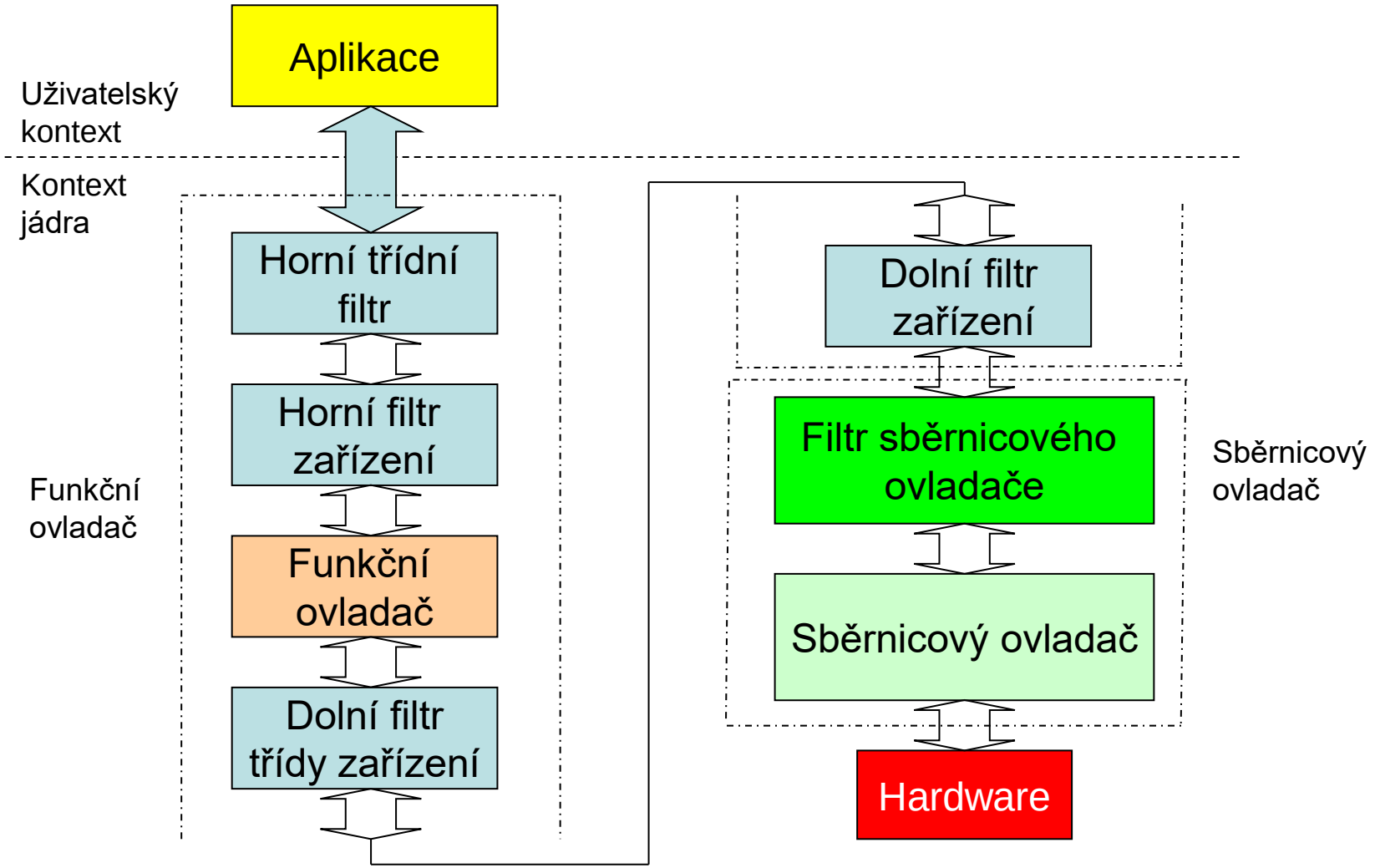
Rozdělení ovladačů dle kontextu

- Kernel Drivers
  - Jsou ovladači v kontextu jádra
  - Mohou přistupovat přímo na hardware a využívat přímého přístupu do paměti
  - Složitější vývoj a ne tak komfortní ladění
- User Mode Drivers
  - Jsou ovlači v uživatelském kontextu
  - Mají formu DLL knihovny
  - Nacházejí v horních patrech hierarchie ovladačů pro dané zařízení
  - Tento typ driverů využívají např. USB zařízení

Rozdělení ovladačů dle funkce

- Sběrnicové ovladače (Bus Drivers)
  - Obsluhují řadiče sběrnic, sběrnicové adaptéry nebo můstky
  - Příkladem jsou sběrnicové ovladače pro PCI, SCSI a USB
  - Najdeme je na nejnižší úrovni v hierarchii
  - Typicky zajišťují enumeraci (identifikaci a evidenci) zařízení připojených na sběrnici a konfiguraci těchto zařízení
- Funkční ovladače (Function Drivers)
  - Ovladač konkrétního typu zařízení (myš, tablet, …)
  - Obvykle poskytován výrobcem zařízení
- Filtry (Filter Drivers)
  - "průchozí" ovladače vkládané mezi ovladače
  - Používají se pro logovací nebo bezpečnostní funkce

Hierarchie ovladačů



Vývoj ovladačů

- Windows Driver Model
  - Původní, k dispozici od Windows 98
  - Knihovní podpora pouze základních funkcionalit
  - Typické funkcionality ovladače je nutno implementovat v každém ovladači znovu a znovu
  - Náročné na vývoj a znalosti
- Kernel-Mode Driver Framework (KMDF)
  - Knihovna, která poskytuje sadu funkcí na vyšší úrovni, které zjednodušují vývoj ovladačů
  - Typické funkcionality jsou implementovány v knihovně
  - Knihovna využívá objektový přístup
  - Implementace v jazyce C
  - Snažší vývoj, zvláště pro začátečníky

Objektový přístup (princip)

Jedná se o objektový přístup ve smyslu objektového programování.
Implementován v jazyce C.

Instanční proměnné jsou ve strukturách

```
typedef struct {
    int pocet;
    void (*clean)(PMY_CLASS obj);
} MY_CLASS, *PMY_CLASS;
```

Metody

```
void set(PMY_CLASS obj, int value) {
          obj->pocet = value;
}
```

Destruktor dealokuje instanční proměnné typu reference na objekt

Nová instance třídy

```
PMY_CLASS my_class_create(PMY_CLASS obj, int value, void (*clean)(PMY_CLASS obj)()) {
          PMY_CLASS obj = (PMY_CLASS)calloc(1, sizeof(MY_CLASS));
          return obj;
}
```

Zrušení instance třídy

```
void my_class_destroy(PMY_CLASS *obj) {
          (*obj)->clean(*obj);        // volání callbacku clean pro výmaz referencí
          free(*obj);
          *obj = NULL;
}
```

Metody připojování periferií

BI-MPP

Přednáška 10

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Ovladače ve Windows (pokračování předchozí přednášky)

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- WDK Documentation, Windows Drivers Kits, Microsoft Corporation, 2009.

Skutečná práce s objekty

WdfObjectCreate - vytvoření objektu
alokace základní datové struktury instance třídy objekt
WdfObjectDelete – zrušení (dealokace) objektu

Každý objekt má počítadlo odkazů (referencí), které se při vytvoření objektu nastaví na jedničku. Vytváříme-li odkazy na objekt v jiných objektech, je nutno programově zvyšovat počítadlo odkazů funkcí **`WdfObjectReference`**, případně snižovat funkcí **`WdfObjectDereference`**, pokud není odkaz dále používán.

Události

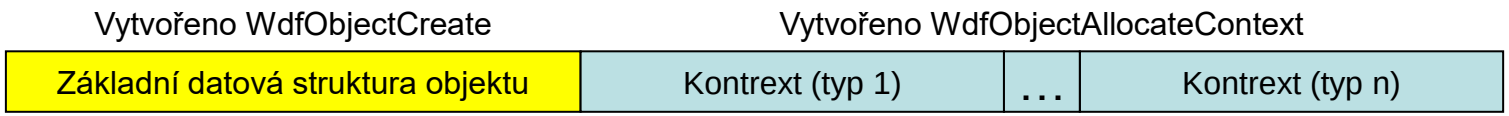
`EvtCleanupCallback` – musí uvolnit veškeré odkazy na daný objekt pomocí funkce **`WdfObjectDereference`** tak, aby počet odkazů dosáhl nuly. Je volán při požadavku na smazání objektu funkcí **`WdfObjectDelete.`**

`EvtDestroyCallback` – volán pokud počítadlo odkazů dosáhlo nuly, tj. těsně před dealokací objektu. Funkce by měla uvolnit veškeré alokované paměťové bloky vázané na daný objekt.

Vytvoření instančních proměnných objektu

`WdfObjectAllocateContext` – alokuje paměťový blok o velikosti struktury, která obsahuje instanční proměnné. Velikost se předává v parametru funkce. Je-li takových struktur více, pak je možné tuto funkci volat vícekrát pro jeden objekt.

`WdfObjectGetTypedContext` – vrací ukazatel na požadovaný kontext.



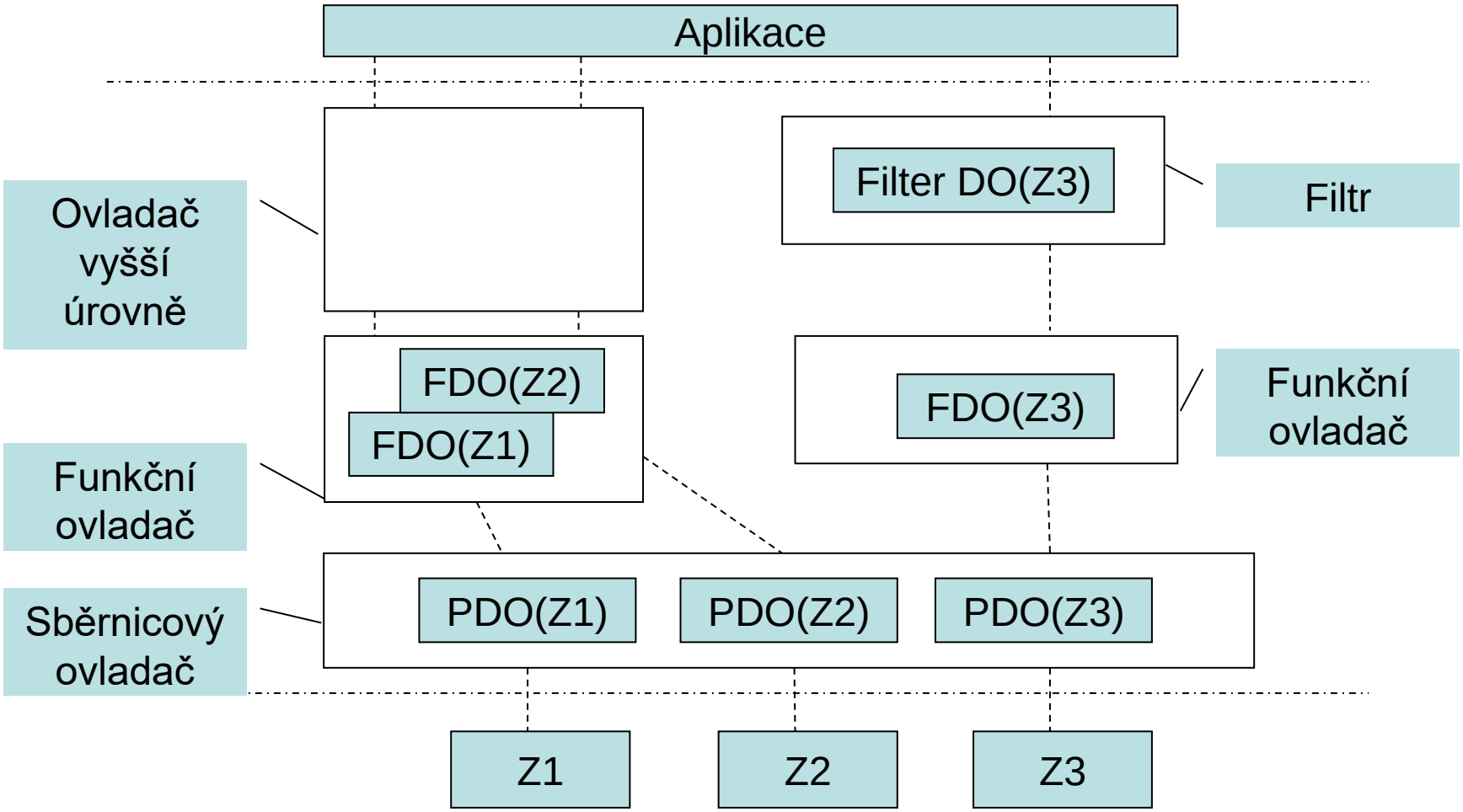
Typy objektů pro zařízení

- PDO (Physical Driver Object) – reprezentuje zařízení na sběrnici u sběrnicových ovladačů
- FDO (Funcional Driver Object) – reprezentuje zařízení u funkčních ovladačů
- Filter DO (Filter Driver Object) – reprezentuje zařízení u filtrů

Objekt ovladače (Driver Object)

- Reprezentuje ovladač (strojový kód ovladače) nahraný do paměti nahraný do paměti).
- Tento objekt vytváří IO manager při nahrání ovladače a předává se jako parametr do vstupního bodu ovladače (funkce DriverEntry).
- Aplikací funkce **`WdfObjectAllocateContext`** se alokuje paměťový prostor pro proměnné, které jsou společné pro všechna zařízení obsluhovaná tímto ovladačem

Objekty v hierarchii ovladačů



Vstupní bod ovladače (EntryPoint)

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT  DriverObject,
                     IN PUNICODE_STRING  RegistryPath)
{
    WDF_DRIVER_CONFIG  config;
    NTSTATUS  status;
    WDF_OBJECT_ATTRIBUTES  attributes;

    WDF_OBJECT_ATTRIBUTES_INIT(&attributes);
    WDF_DRIVER_CONFIG_INIT(&config, EvtDriverDeviceAdd);


    status = WdfDriverCreate(DriverObject, RegistryPath,
                 &attributes, &config, WDF_NO_HANDLE);
        return status;
}
```

Callback volaný při detekci nového zařízení daného typu

Inicializace objektu ovladače. Funkce poskytnutá frameworkem, nutno volat v každém ovladači

EvtDriverDeviceAdd callback

- Volán kdykoliv je přidáno nové zařízení daného typu
- Musí vytvořit FDO pro dané zařízení voláním WdfDeviceCreate
- Vytvořit interfacy ovladače
- Zviditelnit ovladač aplikacím (pokud je požadováno)
- Vytvoření V/V front

Zviditelnění ovladače pro aplikace (přes symbolický link)

```
PCWSTR deviceFilenameStr = L"\\DosDevices\\mpp";

RtlInitUnicodeString(&deviceFilename,
                     deviceFilenameStr);

status = WdfDeviceCreateSymbolicLink(
                     device,
                     &deviceFilename);

if (!NT_SUCCESS(status)) {
        return status;
}
```

V aplikaci

```
h = CreateFile("\\\\.\\mpp", …);
```

Zviditelnění ovladače pro aplikace (přes device interface)

- V ovladači zavolat funkci `WdfDeviceCreateDeviceInterface` a vytvořit (přiřadit) typ rozhraní (myšleno rozhraní aplikace-ovladač) k právě připojenému zařízení
- V aplikaci funkcemi SetupDi…nalézt požadované zařízení dle typu rozhraní a získat souborovou cestu k tomuto zařízení (DevicePath)
- Funkcí CreateFile otevřít ovladač, získat maipulační číslo (handle) a souborovými funkcemi zařízení ovládat

Vytvoření rozhraní (Device Interface)

```
status = WdfDeviceCreateDeviceInterface(device,
              &GUID_DEVINTERFACE_FIT, NULL);
if (!NT_SUCCESS (status)) {
       return status;
}
```

V ovladači se při připojení nového zařízení (konkrétně v callbacku `EvtDriverDeviceAdd`) funkcí `WdfDeviceCreateDeviceInterface` vytvoří rozhraní požadovaného typu (v našem případě reprezentovaného `GUID_DEVINTERFACE_FIT`).

Fakticky se jedná o přiřazení GUID typu rozhraní a jeho záznam do systémových registrů (Registry). Můžeme použít standardní GUID např. `GUID_DEVINTERFACE_DISK`, pak ale musí ovladač implementovat funkce CreateFile, ReadFile, WriteFile, DeviceIoControl, CloseHandle tak, jak standardně požadováno pro zařízení typu DISK. Můžeme si také vytvořit vlastní GUID (jako naše `GUID_DEVINTERFACE_FIT`) a funkcionalitu definovat sami

Detekce zařízení pro požadovaný typ rozhraní

```
// Vytvoření seznamu všech připojených zařízení
// počet zařízení lze specificky omezit vhodnými parametry funkce)
HDEVINFO hdi = SetupDiGetClassDevs(NULL, NULL, NULL,
    DIGCF_ALLCLASSES | DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);

// Vyhledání všech zařízení s rozhraním daného typu

for(i = 0; TRUE; i++) {

          int r = SetupDiEnumDeviceInterfaces(hdi, NULL,
                        &GUID_DEVINTERFACE_FIT, i, &interf);
          if (!r) {
                    int err = GetLastError();
                    if (err == ERROR_NO_MORE_ITEMS) {
                              break;
                    }
                    // report an error !
                return;

            }
          // funkcí SetupDiGetDeviceInterfaceDetail zjisti
          // detailnější informace včetně DevicePath
}
// Použij DevicePath ve funkci CreateFile k získání
// manipulačního čísla (handle)
```

Zpracování V/V požadavků (vytvoření fronty)

```
WDF_IO_QUEUE_CONFIG  ioQueueConfig;
WDFQUEUE  hQueue;

NTSTATUS EvtDriverDeviceAdd (
    IN WDFDRIVER  Driver,
    IN PWDFDEVICE_INIT  DeviceInit
          ) {
    …
          WDF_IO_QUEUE_CONFIG_INIT_DEFAULT_QUEUE(
                    &ioQueueConfig, WdfIoQueueDispatchSequential);

          ioQueueConfig.EvtIoDefault = EvtIoDefault;
          ioQueueConfig.EvtIoRead = EvtIoRead;
          ioQueueConfig.EvtIoWrite = EvtIoWrite;

          status = WdfIoQueueCreate(device,
                    &ioQueueConfig, WDF_NO_OBJECT_ATTRIBUTES,
                              &hQueue);
          if (!NT_SUCCESS (status)) {
                    return status;
          }
    …
    return status;
}
```

Požadavky budou zpracovávány sekvenčně

Callback pro operaci ReadFile

Callback pro operaci WriteFile

Callback pro zpracování požadavku EvtIoRead a EvtIoWrite

```
void EvtIoRead(IN WDFQUEUE  Queue, IN WDFREQUEST  Request,
      IN size_t  Length) {
    //
    // … zpracování požadavku bude zde …
    //
    // signalizace, že požadavek byl zpracován
    WdfRequestComplete(Request, STATUS_SUCCESS);
}

void EvtIoWrite(IN WDFQUEUE  Queue, IN WDFREQUEST  Request,
     IN size_t  Length)
{
    //
    // … zpracování požadavku bude zde …
    //
    // signalizace, že požadavek byl zpracován
    WdfRequestComplete(Request, STATUS_SUCCESS);
}
```

Metody připojování periferií

BI-MPP

Přednáška 11

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- BlueTooth
- Integrace do Linuxu
- Integrace do Windows
- Programová obsluha
- Sériová komunikace

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2

BlueTooth

- Bezdrátová komunikace v pásmu 2,4GHz
- Průběžné a rychlé přeladďování v průběhu vysílání (Frequency Hopping Spread Spectrum) pro větší odolnost vůči rušení na jedné frekvenci
- Řeší všechny OSI vrstvy a po aplikační
- Určeno pro připojování blízkých zařízení (mobilní telefony, handsfree, herní ovladače, GPS, …)
- Je to do jisté míry lepší náhrada IrDA (nepotřebuje fyzické směrování vysílače a přijímače)
- Specifikace BlueTooth 1.0, 1.1, 1.2, 2.1+EDR (Enhanced Data Rate)

BT Profily

- Určují mapování BT zařízení
- Příklady profilů
  - SPP (sériový port)
  - Headset (zvukovka, ovládání hlasitosti)
  - A2DP (přenos audia ve vysoké kvalitě)
  - LAP (LAN Access)
  - PAN (Personal Area Networking)
  - OBEX (Object EXchange)

BlueTooth in Linux

hcitool scan

```
Scanning ...
        00:0B:CE:00:95:E0   BlueCar
        00:21:08:D3:7C:CF   Mirek
```

rfcomm connect 0 00:0B:CE:00:95:E0

```
Zařízení je přístupné na: /dev/rfcomm0
```

Párování

- Zajišťuje jednoznačné přiřazení dvou BT zařízení pro vzájemnou komunikaci
- Kdo se chce připojit k jinému zařízení musí znát jeho párovací klíč
- U párování mobilních telefonů se po žádosti o připojení objeví na telefonu pořadavek na zadání párovacího klíče a stejný klíč je nutné zapsat do formuláře na straně např. PC

Operace pro manipulaci se znakovými zařízeními

- int open(char* dev, int oflag);
- int read(int fd, char* buf, int size);
- int write(int fd, char* buf, int size);
- int ioctl(int fd, int cmd, long arg);
- void close(int fd);

Příklad příjmu dat z BT zařízení na Linuxu

```
int main() {
        int r, h, i = 0;
        char buf[1000];
        struct termios tio;
        h = open("/dev/rfcomm0", O_RDWR | O_NOCTTY );

        while (1) {
                printf("reading %d ...", i++);
                fflush(stdout);
                r = read(h, buf, 1000);
                printf(" done.\n");
                if (r < 0) {
                        printf("Error: reading line\n");
                        return 1;
                }
                if (r > 0) {
                        *(buf + r) = '\0';
                        printf("%s\n", buf);
                }
        }
        close(h);
 }
```

IOCTL pro tty

```
struct termio {
        unsigned short c_iflag;         /* vstupní příznaky */
        unsigned short c_oflag;         /* výstupní příznaky */
        unsigned short c_cflag;         /* řídící příznaky */
        unsigned short c_lflag;         /* lokální příznaky */
        unsigned char c_line;           /* způsob zpracování řádky */
        unsigned char c_cc[NCC];        /* řídící znaky */
} tio;
```

Ignoruj break, ignoruj paritu a ignoruj znak <cr>

```
...
tio.c_iflag = IGNBRK | IGNPAR | IGNCR;
tio.c_oflag = 0;
tio.c_cflag = CS8 | B115200;
tio.c_lflag = ICANON;
...
```

Rychlost 115200 baudů

Kanonický režim

Získání parametrů sériové linky

```
ioctl(fd, TCGETS, &tio);
```

Nastavení parametrů sériové linky

```
ioctl(fd, TCSETS, &tio);        tcsetattr(fd, TCSANOW, &tio)
```

Blokující a neblokující operace

- Pokud je operace read blokující, po zavolání čeká dokud nepřijde požadovaný počet znaků
- Pokud je operace write blokující, čeká dokud se podaří nějakou část dat odeslat
- Neblokující operace se vrací hned. Pokud nelze operaci provést indikuje funkce chybu a errno je nastaveno na EAGAIN.

Kanonický vstup

- Při příjmu se detekují konce řádků
- Nastavením (termios) je možno natavit detekci na různých typů zakončení řádků (<cr>, <cr><lf>, <lf>)
- Operace read se úspěšně vrátí pokud je do bufferu umístěn požadovaný počet znaků nebo je dosaženo konce řádku

Nekanonický vstup

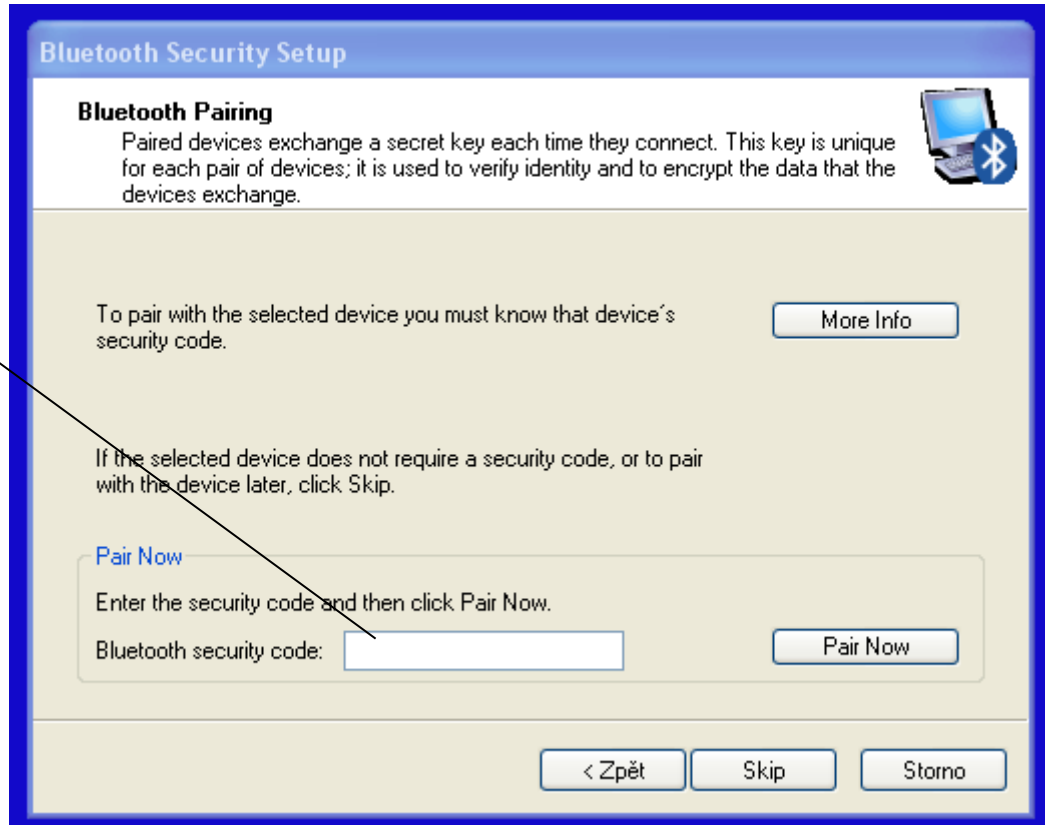
- Konce řádku nejsou detekovány
- Funkce read se vrátí pokud je přijat požadovaný počet znaků, je mezi dvěma přijatými znaky delší prodleva než parametr VTIME, je přijato VMIN znaků.

Select

- Funkce select dovoluje sledovat více sériových zařízení. Pokud nějaké zařízení má k dispozici data, nebo je schopno vysílat, dojde k návratu z funkce. Jinak je funkce blokuje (čeká).
- Funkce dovoluje předem testovat, zda dané zařízení bude pro požadovanou operaci blokovat.
- Výhodou je, že funkce poskytuje timeout, tak se lze vyhnout uváznutím programu při kanonickém vstupu, který neposkytuje timeouty pro funkci read. K tomu např. může dojít, u zařízení, které má chybu v protokolu nebo dojde k chybě při přenosu.

Žádost o spárování telefonu

Zadejte klíč zde. Stejný klíč musíte zadat také na telefonu



Služby poskytované mobilním telefonem

Datové přenosy COM port

Modem (vytáčené připojení)

Headset (emulace)

Mapování BT zařízení ve Windows

- Zařízení s SPP profilem se mapují jako COM porty
- Zařízení, které dovolují audio přenosy se mapují jako BT zvukové karty a je možné audio data získat například z Windows Multimedia API

Komunikace s BT zařízení na portu COM6

Získání manipulačního čísla sériové linky

```
handle = CreateFile("\\\\.\\COM6", GENERIC_READ |
       GENERIC_WRITE, 0, NULL, OPEN_EXISITING, 0, NULL);

if (handle == INVALID_HANDLE_VALUE) {
       ... informuj o chybě ...
       return;
}
```

Uzavření zařízení

```
CloseHandle(handle);
```

Čtení a zápis dat z BT zařízení

```
char buffer[100];
int size;

if (!ReadFile(handle, buffer, 100, &size, NULL)) {
        ... Informuj o chybě ...
        return;
}
```

Délka

Délka přečtených dat

```
char buffer[100];
int size;

if (!WriteFile(handle, buffer, 100, &size, NULL)) {
        ... Informuj o chybě ...
        return;
}
```

Délka

Délka skutečně zapsaných dat

Timeouty

Získání nastavení timeoutů

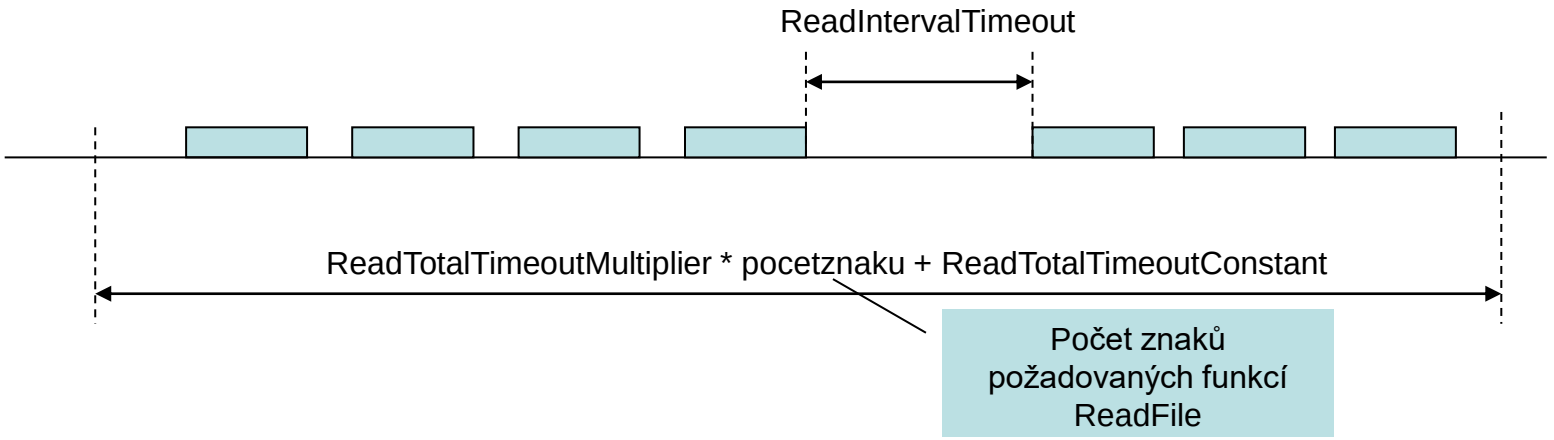
```
COMMTIMEOUTS timeouts;

GetCommTimeouts(handle, &timeouts);
```

Nastavení timeoutů

```
SetCommTimeouts(handle, &timeouts);
```

Struktura COMMTIMEOUTS

```
typedef struct _COMMTIMEOUTS {
DWORD ReadIntervalTimeout;
DWORD ReadTotalTimeoutMultiplier;
DWORD ReadTotalTimeoutConstant;
DWORD WriteTotalTimeoutMultiplier;
DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS;
```



Asynchronní operace čtení a zápisu z BT zařízení

```
OVERLAPPED overlapped;
memset(&overlapped, 0, sizeof(overlapped));

hEvent = CreateEvent(NULL, TRUE, FALSE, "Event");
Overlapped.hEvent = hEvent;

handle = CreateFile("\\\\.\\COM6",GENERIC_READ | GENERIC_WRITE,
       0, NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL);
 if (handle == INVALID_HANDLE_VALUE) {
        ... Informuj chybu ...
        return;
 }

r = ReadFile(handle, buffer, 100, &size, &overlapped);

... dělej něco jiného ...

WaitForSingleObject(hEvent, 5000);
```

Manual Reset

Signalling state

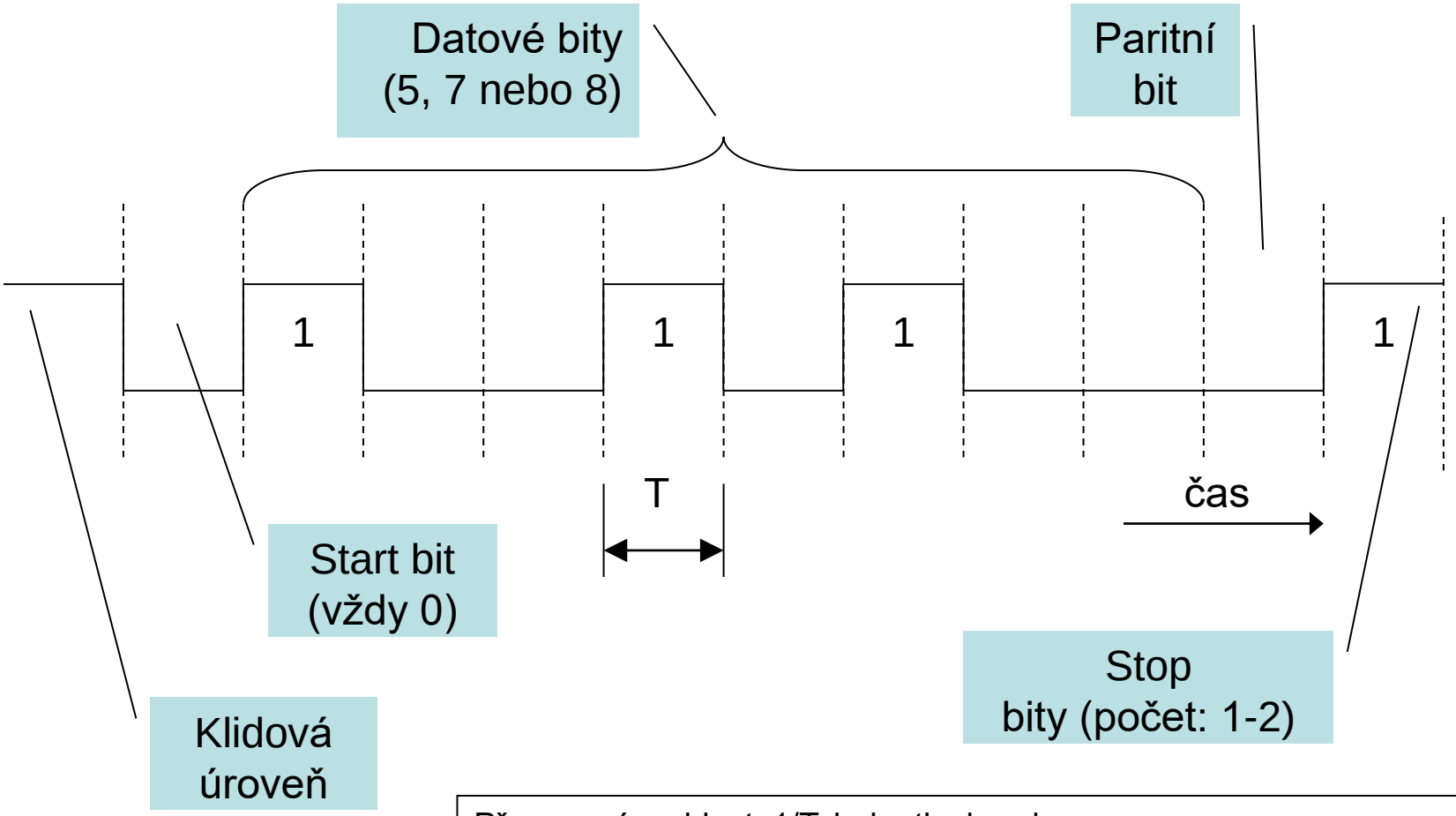
Timeout

Čeká na dokončení operace

Sériový port

- Historicky starší rozhraní než počítače PC
- Určeno pro připojení modemu nebo terminálu
- Asynchronní sériový komunikační protokol
- Normalizováno RS232 standard nebo V.24 ITU-T standard.
- Datové signály jsou doprovázeny signály pro řízení modemu
- Nesymetrická linka, vzdálenost do 15m
- Použití: připojení modemu, komunikace mezi počítači, připojování specializovaných zařízení k PC

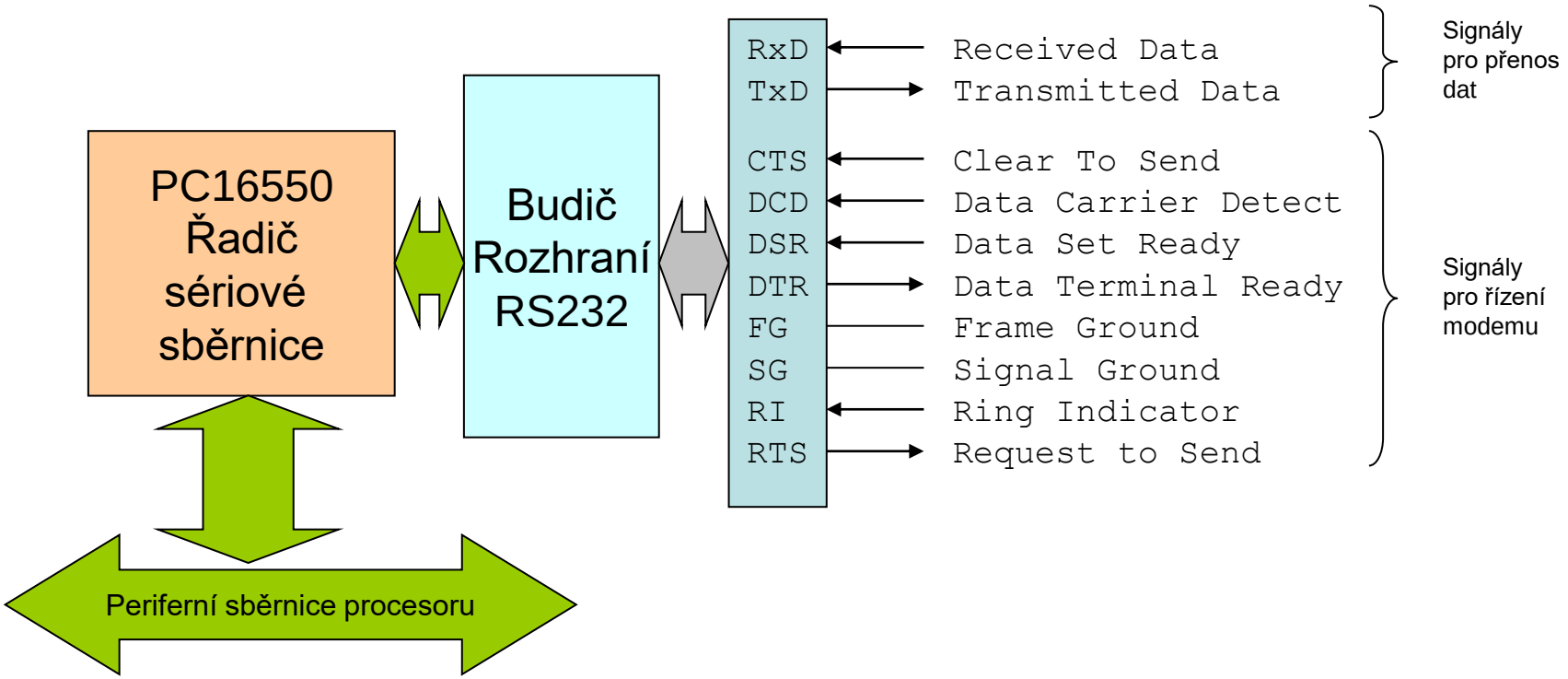
Protokol sériového rozhraní



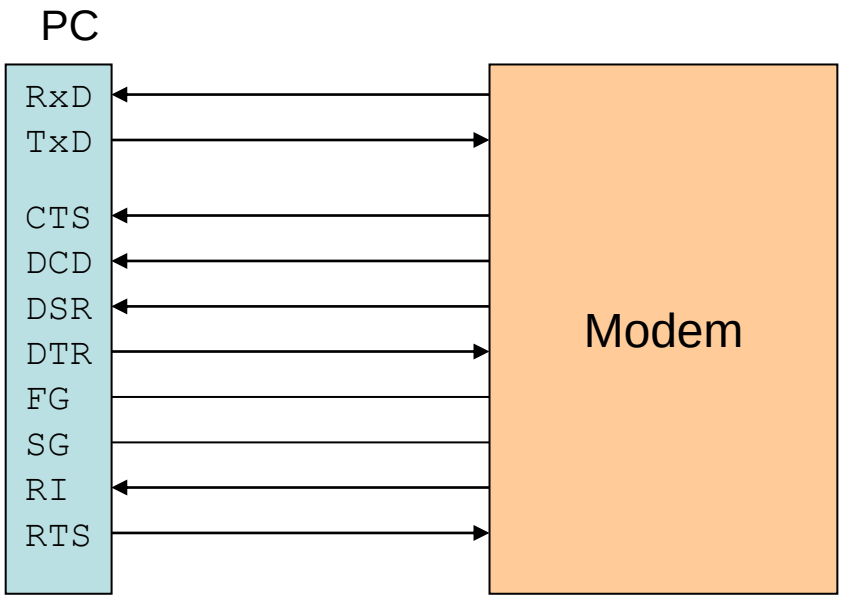
Přenosová rychlost: 1/T, jednotka baud

Typické rychlosti: 2400, 4800, **9600**, 19200, 38400, 57600, **115200**

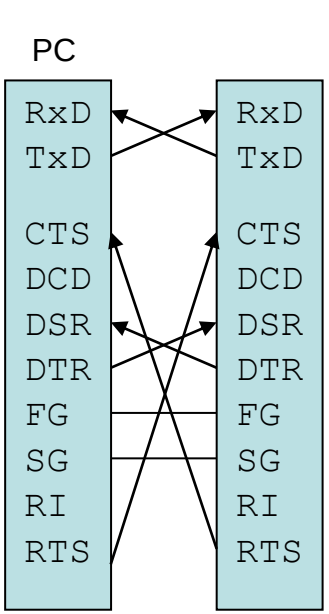
Sériové rozhraní



Připojení modemu



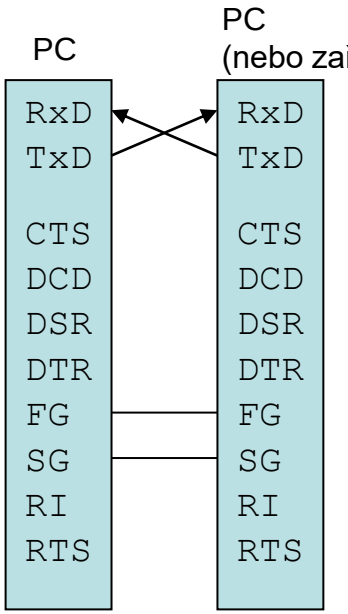
Nulový modem



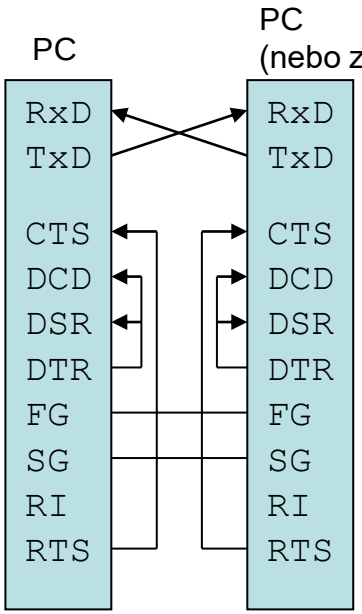
Řízení toku
Hardware

Doporučení Microsoftu

zdroj
http://support.microsoft.com/kb/q142324/



Řízení toku
None nebo
Software



Řízení toku
None nebo
Software

Metody připojování periferií

BI-MPP

Přednáška 12

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Video for Linux
- Windows Multimedia API
- XAudio2

Literatura

- Gook, M.: Hardwarová rozhraní – Průvodce programátora. Computer Press, Brno 2006. ISBN 80-251-1019-2
- LINUX MEDIA INFRASTRUCTURE API. LinuxTV Developers 2009-2011. (http://linuxtv.org/downloads/v4l-dvb-apis/)
- Windows Multimedia Functions (http://msdn.microsoft.com/en-us/library/ms925318.aspx)

Video for Linux (v4l2)

- Linuxový standard pro přístup k video zařízením
- Zdroj: *Schimek, M., H.: Video for Linux Two API Specification, Revision 0.24.*
- Rozhraní definováno nad souborovými operacemi open, read, write, ioctl, close.
- Pro rychlejší výměnu dat je možno přímé mapování bufferů do paměťového prostoru aplikace (mmap).
- Pro řízení video zařízení je definována sada funkcí řízených přes ioctl

Video zařízení v Linuxu

```
/dev/video          # defaultní zařízení
                    # typicky link na
                    # /dev/video0
```

```
# jednotlivá video zařízení

/dev/video0
/dev/video1
/dev/video2
/dev/video3
...
```

Otevření video zařízení

```
int fd = open("/dev/video0");
if (fd < 0) {
      // signalizace chyby

}
```

Proměnná fd (file descriptor) je dále užívána jako odkaz na otevřené zařízení

Zjištění informací o video zařízení

```
struct v4l2_capability cap;

If (ioctl(fd, VIDIOC_QUERYCAP, &cap) < 0 {
     // signalizace chyby
}
```

Cap obsahuje položku capabilities s hodnotami:

- V4L2_CAP_VIDEO_CAPTURE - má rozhraní pro snímání obrazu (capture)
- V4L2_CAP_VIDEO_OVERLAY – má rozhraní přímé ukládání obrazu do video paměti grafické karty
- V4L2_CAP_AUDIO – zařízení má audio vstup/výstup

Zjištění počtu a nastavení video vstupu

VIDIOC_ENUMINPUT zjišťuje počet video vstupů. Argumentem je `struct v4l2_input`

VIDIOC_S_INPUT nastavuje požadovaný vstup. Argumentem je `int*`

Metody čtení/zápisu dat z/do video zařízení

- Read/write
  - Metoda užívá standardních read/write operací. Může mít přidanou režii pro kopírování dat mezi user space a kernel space, pokud driver nepoužívá DMA
- Streaming (paměťově mapované buffery)
  - Používá funkce mmap pro mapování fyzické paměti do virtuálního adresního prostoru
  - Video buffery jsou alokovány ve fyzické paměti
- Streaming (uživatelské ukazatele)
  - Video buffery jsou alokovány v aplikaci (user space)
  - Do driveru se předávají pouze ukazatele na buffery

Streaming (paměťově mapované buffery)

- Ioctl VIDIOC_REQBUFS se použije pro alokaci video bufferů ve fyzické paměti na straně ovladače. Argumentem je požadovaný počet bufferů. Počet alokovaných bufferů může být menší než požadovaný
- Vytvoří se pole popisovačů bufferů. Popisovač obsahuje adresu počátku bufferu v user space a jeho délku.
- Pole se vyplní opakovaným voláním ioctl VIDIOC_QUERYBUF, které dodá délku bufferu. Adresu počátku bufferu zjistíme funkcí mmap, která buffer namapuje do aplikace. Adresu doplníme do popisovače
- Aplikace volá VIDIOC_QBUF ioctl (enqueue the buffer), které zařadí buffer do fronty v ovladači a ten ji pak buffer vyplní snímkem. Vyplněný buffer může aplikace převzít voláním VIDIOC_DQBUF ioctl (dequeue the buffer).
- Typicky máme-li 5 bufferů, pak pro všechny voláme VIDIOC_QBUF ioctl, dále zavoláme VIDIOC_STREAMON a pak následuje čtecí smyčka, kde testujeme stav bufferu (VIDIOC_QUERYBUF ioctl) a přejímáme voláním VIDIOC_DQBUF ioctl.

Windows Multimedia API

- API pro přehrávání a snímání audia a videa
- Dnes se nahrazuje jednoduššími a multiplatformními (PC a XBox) rozhraními z DirectX (XAudio2)
- Přehrávání videa i audia

Otevření streamu pro záznam

```
result = waveInOpen(&hWaveIn, deviceId, &pFormat,
            (DWORD_PTR) waveInProc, (DWORD)
             & audio_buffers,
                  WAVE_FORMAT_DIRECT
                | WAVE_MAPPED |
                  CALLBACK_FUNCTION);
if (result != MMSYSERR_NOERROR) {
        return 0;
}
```

Příprava hlavičky bufferu

```
waveInPrepareHeader(hWaveIn, hdr, sizeof (WAVEHDR));
```

Tato funkce připravuje hlavičku bufferu pro to, aby mohl být použit pro ukládání nebo přehrávání audio dat. Tato funkce koncentruje všechny časově náročné inicializace jak na straně audio driveru, tak operačního systému.

Zařazení bufferu do fronty

```
// přidání  bufferu do fronty
waveInAddBuffer(hWaveIn, &audio_buffers.WaveInHdr[i],
sizeof (WAVEHDR));
```

Po naplnění bufferu (indikováno např. voláním callback funkce) aplikace naplní/překopíruje audio data a může znovu ten samý buffer zařadit do fronty funkcí waveInAddBuffer. Toto se děje opakovaně nad množinou třeba 5 bufferů.

Start a stop záznamu

```
// start záznamu

waveInStart(hWaveIn);
```

```
// stop záznamu

waveInReset(hWaveIn);
```

Funkce pro přehrávání

- waveOutOpen
  - Otevření výstupního streamu
- waveOutPrepareHeader
  - Inicializace bufferu
- waveOutWrite
  - Zařezení bufferu do fronty pro přehrávání

Další funkce lze nalézt v MSDN
http://msdn.microsoft.com/en-us/library/ms925318.aspx

XAudio2

- Audio API – součást DirectX
- Podporuje míchání hlasů
  - SourceVoice (zdrojové hlasy)
  - Submix Voice (pomocné míchání uvnitř struktury míchaných hlasů)
  - Master Voice (výsledné míchání a výstup)
- Nepodporuje audio input

Metody připojování periferií

BI-MPP

Přednáška 13

Ing. Miroslav Skrbek, Ph.D.

Katedra číslicového návrhu
Fakulta informačních technologií
České vysoké učení technické v Praze

ZS2021/22

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Agenda

- Přístup k periferiím z virtuálních strojů

Přístup k periferiím z virtuálních strojů

Nativní metody v technologii .NET

Nativní metody jsou napsané v jazyce C nebo C++ a zapouzdřené do DLL knihoven. Jazyky jako je Visual Basic nebo C# mají rozšíření pro volání nativních metod.

Visual Basic

```
Imports System.Runtime.InteropServices

<DllImport("c:\my.dll", CallingConvention:=CallingConvention.Cdecl)> _
Private Shared Function add(a As Integer, b As Integer) As Integer
End Sub

…
c = add(10, 10)
```

C#

```
using System.Runtime.InteropServices;

[DllImport("c:\my.dll")]
private static extern int add(int a, int b);

…
c = add(10, 10)
```

Nativní DLL knihovna pro .NET

použije se C jmenná konvence (naming convention) uvnitř C++

funkce se exportuje z DLL

```
extern "C" {
	__declspec(dllexport) int add(int a, int b) {
		return a + b;
	}
}
```

Nativní metody v technologii JAVA

- Java používá pro volání nativních funkcí speciální rozhraní zvané Java Native Interface (JNI)
- Kromě volání nativních metod, rozhraní dovoluje manipulaci s javovskými objekty a volání metod
- V Linuxu jsou nativní metody zapouzdřeny do sdílených knihoven (lib*.so)
- Ve Windows jsou nativní metody zapouzdřeny do DLL knihoven
- Java poskytuje nástroje pro automatické generování hlavičkových souborů pro jazyk C a C++

Deklarace nativních metod v Javě

```
package mypackage;
class MyClass {
	{	System.loadLibrary("calc");
	}
	public native int add(int a, int b);
}
```

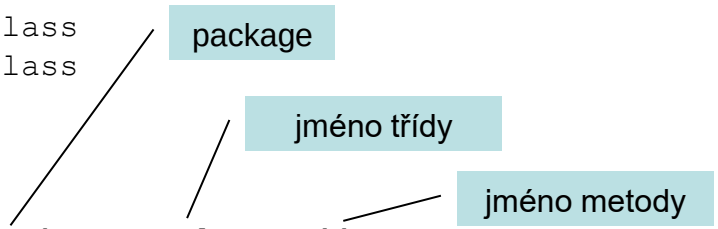
Generování hlavičkových souborů

```
javah –classpath . –jni mypackage.MyClass
```

Předpokládá se, že soubor MyClass.class je umístěn v adresáři ./mypackage

Hlavičkový soubor mypackage_MyClass.h

```
#include <jni.h>
#ifndef _Included_mypackage_MyClass
#define _Included_mypackage_MyClass
#ifdef __cplusplus
extern "C" {
#endif

JNIEXPORT jint JNICALL Java_mypackage_MyClass_add
          (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
}

#endif
#endif
```



Implementace (calc.dll)

```
#include "mypackage_MyClass.h"

JNIEXPORT jint JNICALL Java_mypackage_MyClass_add
  (JNIEnv *env, jobject self, jint a, jint b) {
      return a + b;
}
```

reprezentuje objekt, který obsahuje nativní metodu

Ukazatel na datovou strukturu, která obsahuje ukazatele na funkce JNI

Řetězce v JNI

```
JNIEXPORT jint JNICALL Java_mypackage_MyClass_search
          (JNIEnv *env, jobject self, jchar c, jstring s) {

          int i, len;

          jchar* str = (*env)->GetStringChars(env, s, NULL);

          if (str == NULL) {
                    return NULL;
          }

          len = (*env)->GetStringLength(env, s);

          for(i = 0; i < len; i++) {

                    if (str[i] == c) {

                              (*env)->ReleaseStringChars(env, s, str);

                              return i;

                    }
          }

          (*env)->ReleaseStringChars(env, s, str);

          return -1;

}
```

Rozšíření Pythonu v C/C++

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>

// definice metod
static PyMethodDef MyModuleMethods[] = {
    {"add", add, METH_VARARGS, "add two integers"},
    {NULL, NULL, 0, NULL}
};
// definice modulu
static struct PyModuleDef mymodule = {
    PyModuleDef_HEAD_INIT,
    "mymodule",   /* jméno modulu */
    NULL,
    -1,
    MyModuleMethods
};
```

Pokračování na dalším slidu

Metody a inicializační funkce

```
// Inicializační metoda
PyMODINIT_FUNC PyInit_mymodule(void)
{
    return PyModule_Create(&mymodule);
}

// Metoda ADD
static PyObject *add(PyObject *self, PyObject *args)
{
    int a, b;
    if (!PyArg_ParseTuple(args, “ii", &a, &b))
        return NULL;
    return PyLong_FromLong(a+b);
}
```

Překlad a použití

Vložení CFLAGS pro Python

```
gcc -o mymodule.so `pkg-config --cflags python3` \
      mymodule.c -shared `pkg-config --libs python3`
```

Vložení knihoven pro Python

test.py

```
import mymodule as mm
print(mm.add(3,5))
```

Testování

PYTHONPATH=. python3 test.py