Principy fungování USB a návrhu periferních zařízení

ITP 10.4.2018

Přednáší: Michal Bidlo

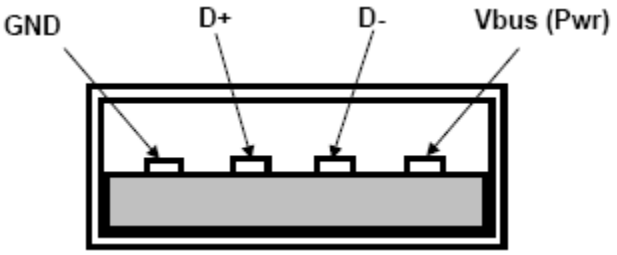
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Obsah

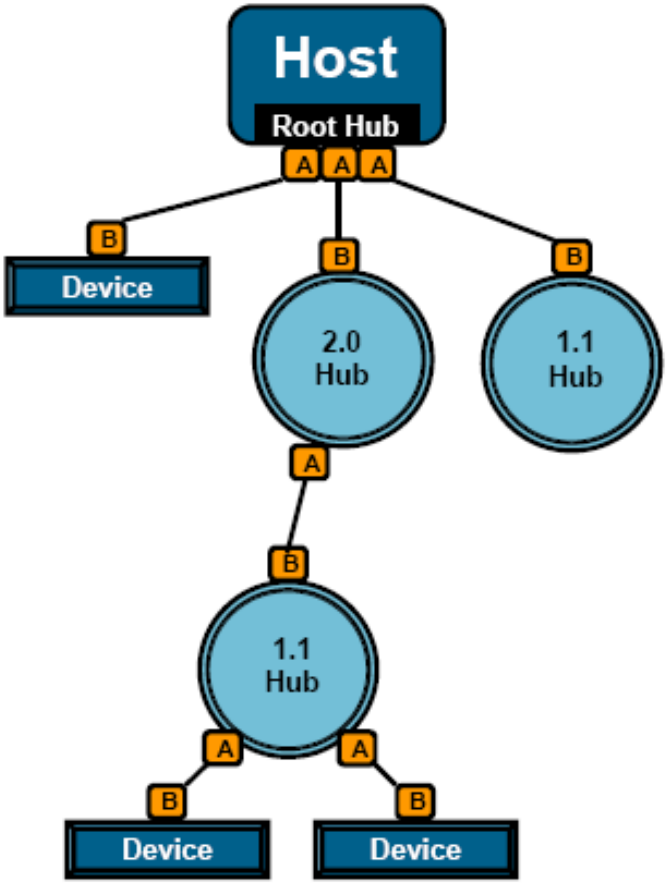
- Rozhraní USB – architektura, princip
  (přehled informací z normy o USB)
- USB modul na mikrokontroléru HCS08 JM60
  (specifikace od výrobce MCU)
- Příklad: firmware emulující USB myš
  (...využívající vývojový kit Freescale DEMOJM)

Část 1: obecně o USB dle normy

Struktura rozhraní USB



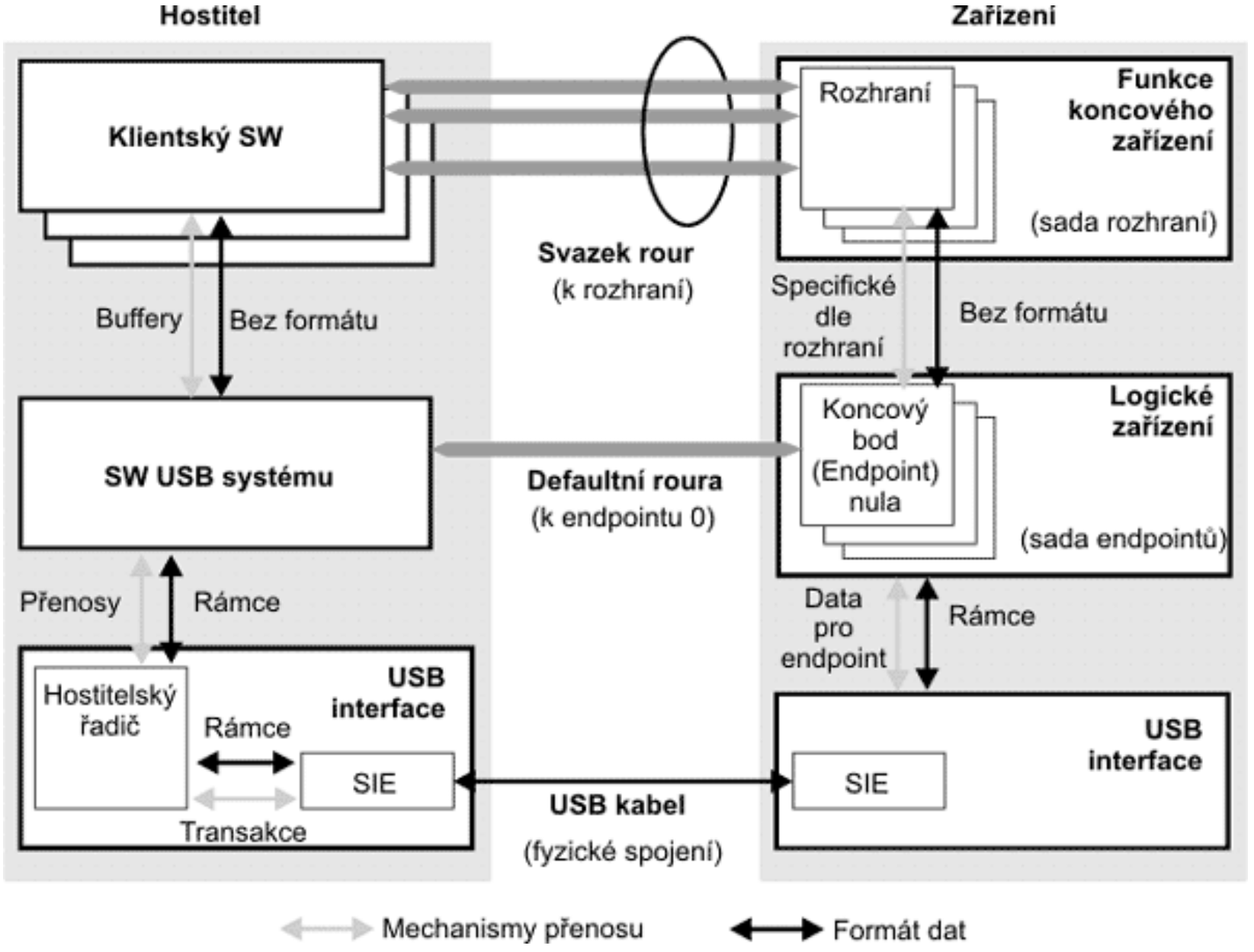
- Asynchronní sériové rozhraní
- Architektura master-slaves (host-devices, počítač-zařízení)
  - Lze připojit až 127 zařízení (pomocí tzv. rozbočovačů – USB hubs)
  - Master řídí připojování a odpojování zařízení, jejich konfiguraci (tzv. enumerace) a dále veškerou komunikaci (přenos dat)
- USB 2.0: tři režimy rychlosti
  - Low-speed (do 1.5 Mbps)
    Full-speed (do 12 Mbps)
    High-speed (do 480 Mbps)



Princip rozhraní USB

- Přenos informací z/do zařízení probíhá na tzv. endpointech (EP) – adresovatelných místech sloužících pro vysílání/příjem dat jako komunikace s počítačem.
  - Příchozí EP (OUT): přenos z počítače do zařízení
  - Odchozí EP (IN): přenos ze zařízení do počítače
- USB zařízení má obvykle několik EP (náš MCU jich má 7)
- Každé USB zařízení musí implementovat obousměrný EP0 pro enumeraci, řídicí a stavové přenosy.
- SW (ovladač) v počítači komunikuje přes tzv. roury (USB pipes), což je asociace s určitým endpointem zažízení.
- Každé zařízení má ustavenu rouru mezi počítačem a EP0.

Princip rozhraní USB: endpointy, roury



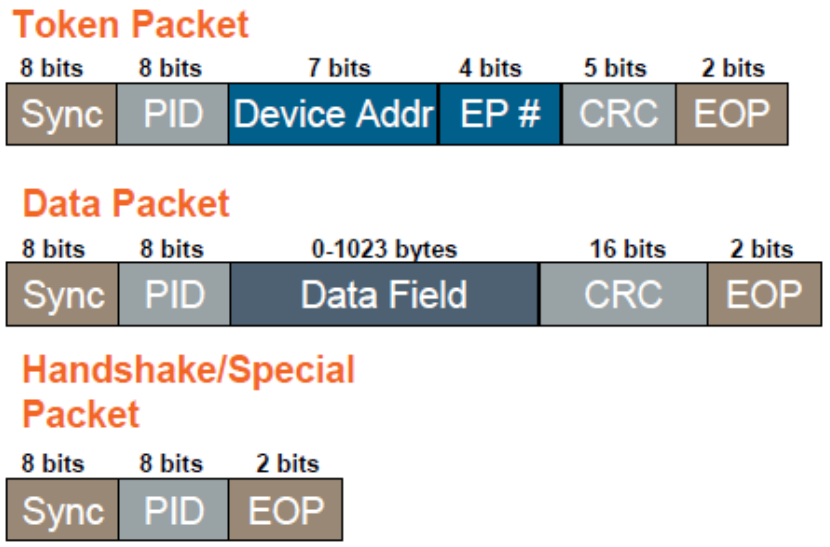
http://hw.cz/Rozhrani/ART1244-USB-2.0---dil-2.html

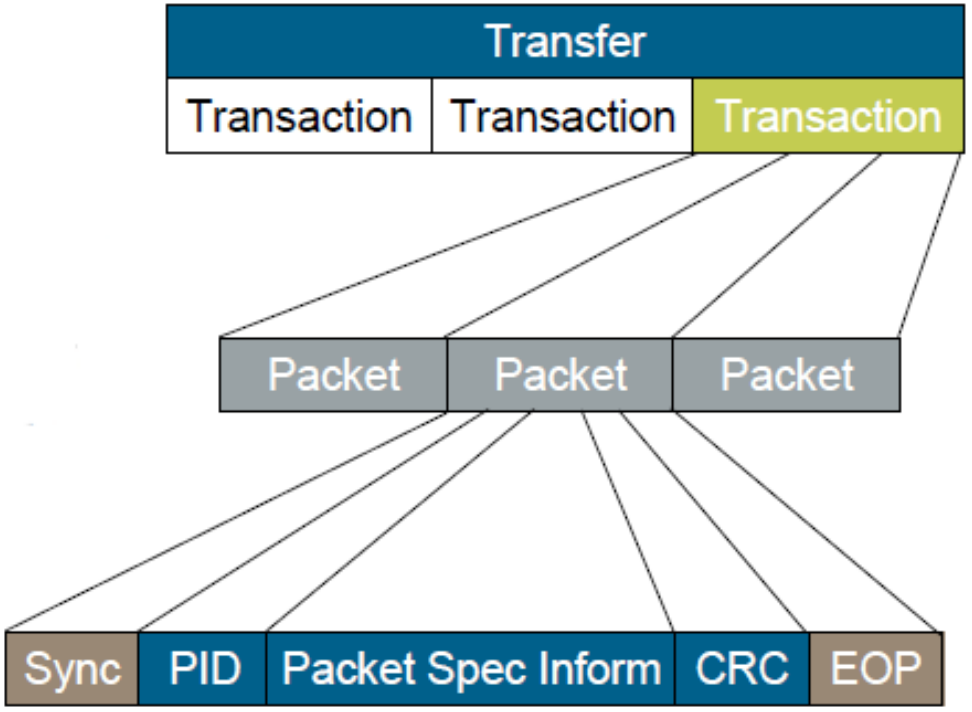
Komunikace na sběrnici USB

- Rozlišujeme 4 typy přenosů na sběrnici USB, které je možné specifikovat pro daný EP (zařízení jich může využívat i více):
- Řídicí přenos (control) – obousměrný počítačem řízený přenos příkazů, stavových informací apod., zásadní při zpracování žádostí v procesu enumerace (viz dále).
- Přerušení (interrupt) – přenos s garantovanou latencí obsluhy, použití hlavně pro přenos stavových informací, znaků nebo dat v definovaných intervalech činnosti zařízení.
- Dávkový přenos (bulk) – nejrychlejší, vybaven protokolem pro detekci a opravu chyb (garantovaný bezchybný přenos), použití pro velké objemy dat – pakety o velikosti až stovek bytů).
- Isochronní (isochronous) – používán při proudových přenosech (audio, video), negarantuje bezchybnost, pakety o velikosti až přes 1000 bytů.

Komunikace na sběrnici USB

- Přenos (transfer) je proces zajišťující kompletní vykonání určitého příkazu od počítače USB zařízením. Je tvořen posloupností transakcí (přenos informace na endpointu). Transakce sestává z tzv. paketů, přesně definovaných struktur, jejichž formát a význam je dán specifikací USB.
- Typy paketů





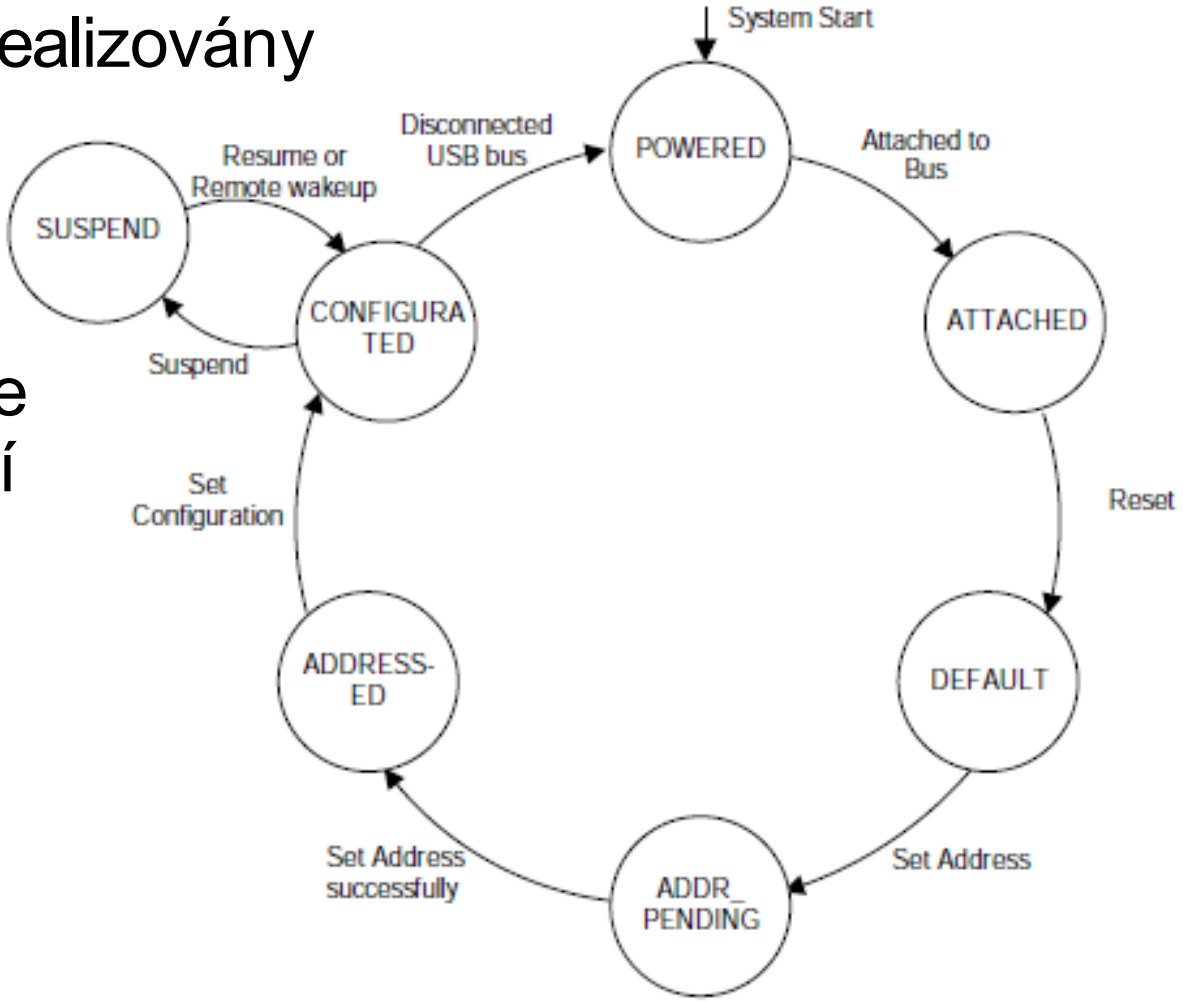
Připojení zařízení k USB, enumerace

- Zařízení musí po připojení reagovat na výzvy počítače (specifikované standardem USB), podle kterých zašle počítači požadované informace.
- Za tímto účelem musí každé zařízení implementovat obousměrný EP0, který realizuje řídicí přenosy.
- Proces enumerace sestává z (řízeno počítačem):
  - resetu zařízení,
  - žádosti o zaslání desktiptoru zařízení na EP0,
  - přidělení adresy (po připojení musí zařízení odpovídat na adresu 0),
  - zaslání dalších deskriptorů (konfigurace, rozhraní,...),
  - po zavedení ovladače je zařízení připraveno k funkci.

Typický stavový cyklus USB zařízen

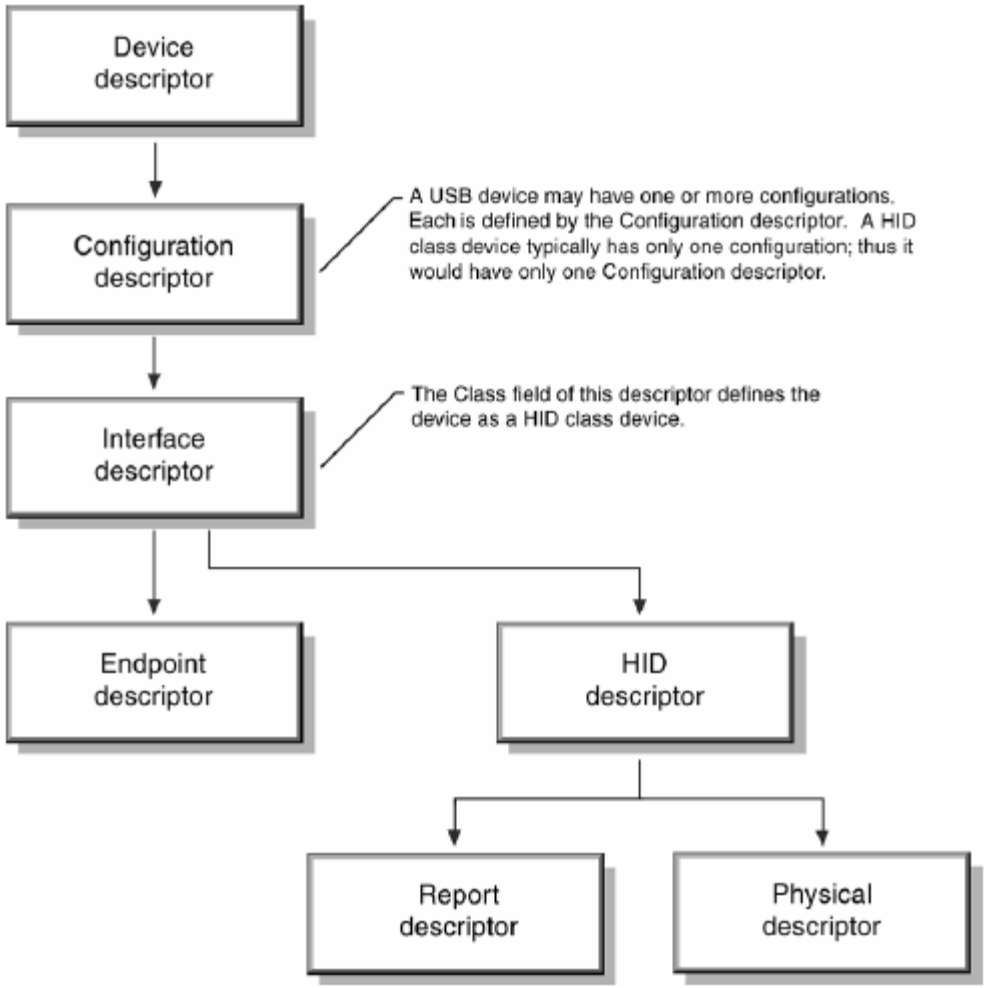
- Proces enumerace je implementován jako určitá sekvence stavů FSM, v rámci kterých jsou realizovány příslušné akce.

- Kromě stavů adresace a konfigurace zařízení zde máme též stav režimu „suspend“, jelikož každé zařízení standardu USB musí podporovat nízko-příkonový režim.



Specifikace USB zařízení

- Každé zařízení je popsáno pomocí tzv. Deskriptorů - přesně definovaných datových struktur obsahujících informace o podporovaném USB rozhraní, funkcích, řízení apod.



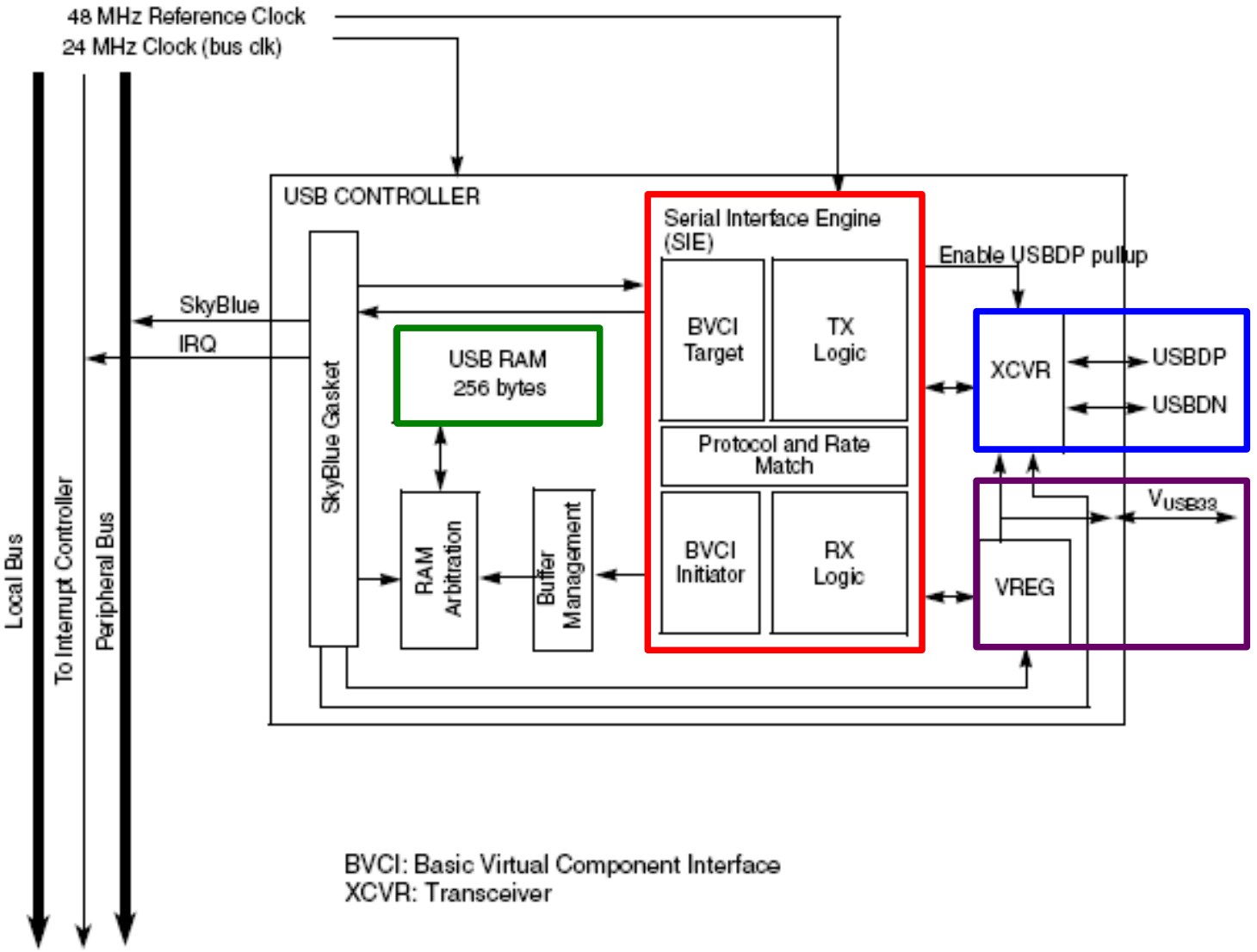

```
/*USB device descriptor structure*/
typedef struct _USB_DEV_DSC
{
    byte bLength;
    byte bDscType;
    word bcdUSB;
    byte bDevCls;
    byte bDevSubCls;
    byte bDevProtocol;
    byte bMaxPktSize0;
    word idVendor;
    word idProduct;
    word bcdDevice;
    byte iMFR;
    byte iProduct;
    byte iSerialNum;
    byte bNumCfg;
} USB_DEV_DSC;

/*USB configuration descriptor structure*/
typedef struct _USB_CFG_DSC
{
    byte bLength;
    byte bDscType;
    word wTotalLength;
    byte bNumIntf;
    byte bCfgValue;
    byte iCfg;
    byte bmAttributes;
    byte bMaxPower;
} USB_CFG_DSC;
```

Část 2: použití USB modulu v MCU dle specifikace od výrobce

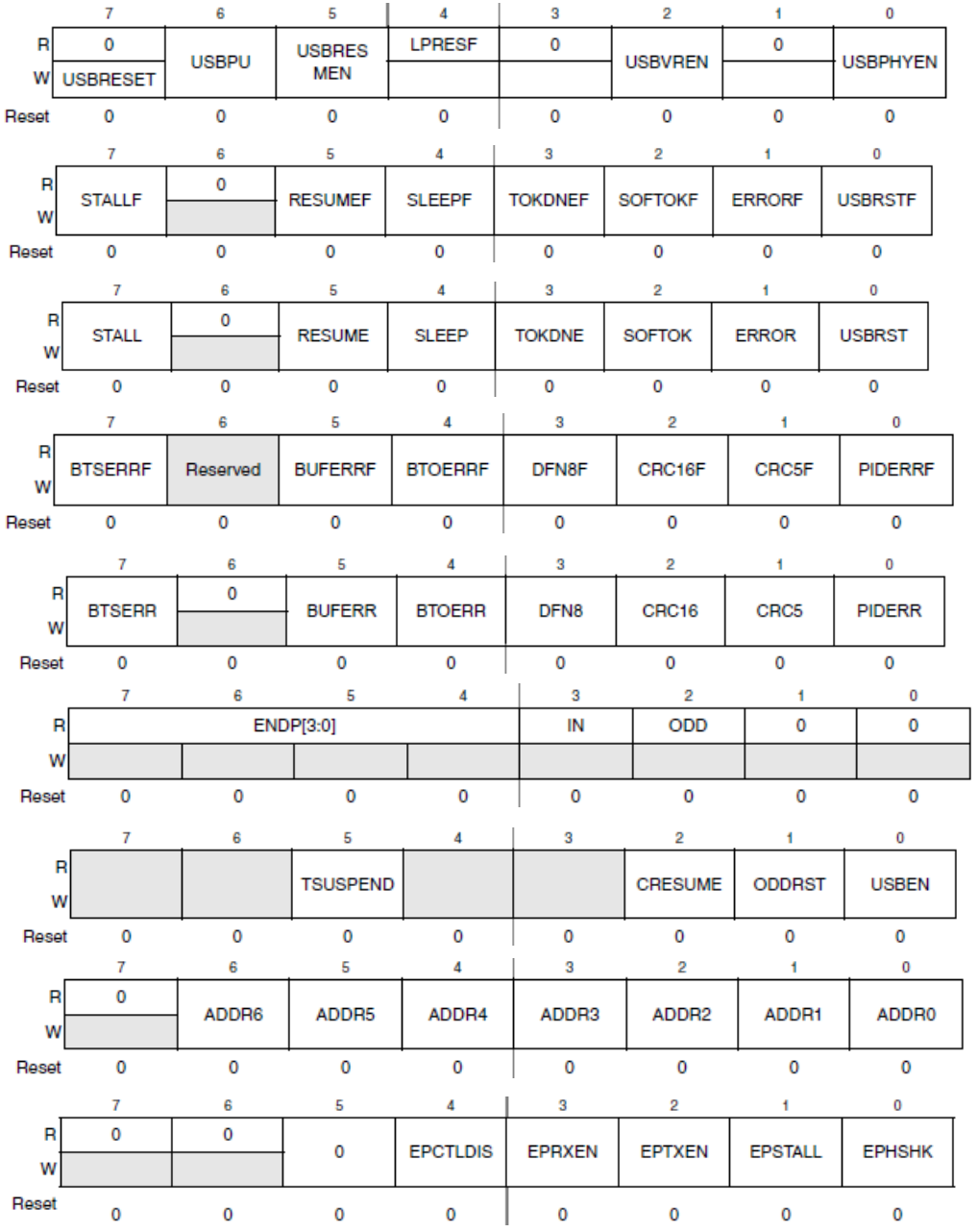
USB mobul na HCS08 JM60

- Klíčovými komponentami pro řízení komunikace s USB zařízením jsou: SIE, XCVR, VREG, USB RAM.



Registry modulu USB na MCU JM60

- Konfigurace a řízení stavu USB modulu se řeší pomocí standardních registrů MCU přístupných prorgamátorovi
  - USB Control Register 0 (USBCTL0)
  - Interrupt Status Reg. (INTSTAT)
  - Interrupt Enable Reg. (INTENB)
  - Error Interrupr Status Register (ERRSTAT)
  - Error Interrupt Enable Register (ERRENB)
  - Status Reg. (STAT)
  - Control Reg. (CTL)
  - Address Reg. (ADDR)
  - Endpoint Control Reg. (EPCTLn)



	7	6	5	4	3	2	1	0
R	0	USBPU	USBRESMEN	LPRESF	0	USBVREN	0	USBPHYEN
W	USBRESET							
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	STALLF	0	RESUMEF	SLEEPF	TOKDNEF	SOFTOKF	ERRORF	USBRSTF
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	STALL	0	RESUME	SLEEP	TOKDNE	SOFTOK	ERROR	USBRST
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	BTSERRF	Reserved	BUFERRF	BTOERRF	DFN8F	CRC16F	CRC5F	PIDERRF
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	BTSERR	0	BUFERR	BTOERR	DFN8	CRC16	CRC5	PIDERR
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	ENDP[3:0]				IN	ODD	0	0
W								
Reset	0	0	0	0	0	0	0	0

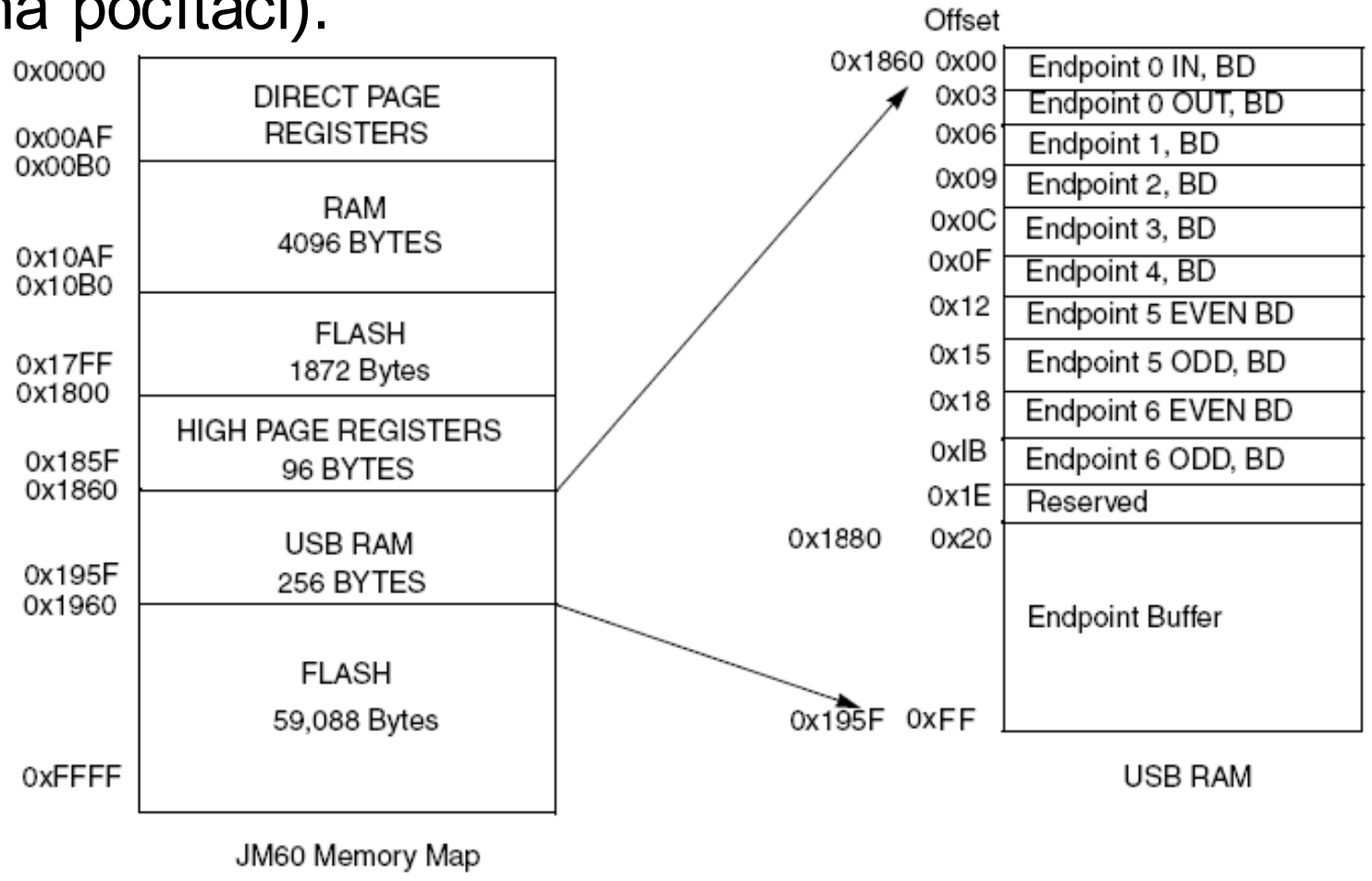
	7	6	5	4	3	2	1	0
R			TSUSPEND			CRESUME	ODDRST	USBEN
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	0	ADDR6	ADDR5	ADDR4	ADDR3	ADDR2	ADDR1	ADDR0
W								
Reset	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
R	0	0	0	EPCTLDIS	EPRXEN	EPTXEN	EPSTALL	EPHSHK
W								
Reset	0	0	0	0	0	0	0	0

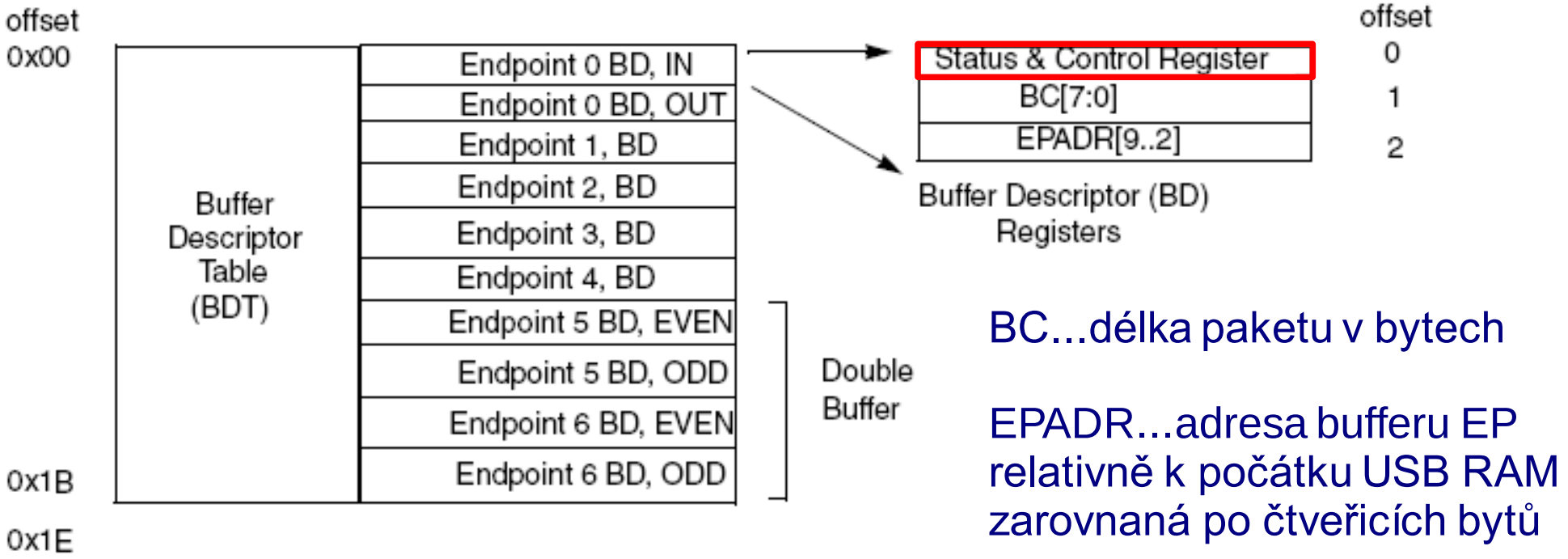
USB RAM

- 256 bytů mapovaných do adresového prostoru 0x1860-0x195F, na jejím počátku jsou registry pro řízení EP (tzv. buffer-deskriptory), od adresy 0x1880 je prostor pro data EP (přímo sem se ukládají data přijatá od počítače nebo vysílaná počítači).



USB RAM: buffer descriptors (BD)

- Každý EP má v USB RAM vyhrazeny 3 byty (BD), které udávají stav, velikost a umístění bufferu EP v USB RAM.



BC…délka paketu v bytech

EPADR…adresa bufferu EP relativně k počátku USB RAM zarovnaná po čtveřicích bytů

Bit	7	6	5	4	3	2	1	0
	OWN	DATA0/1	0 BDTKPID[3]	0 BDTKPID[3]	DTS BDTKPID[1]	BDTSTALL BDTKPID[0]	0	0

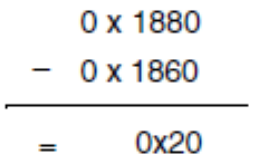
Status and Control Register of BD

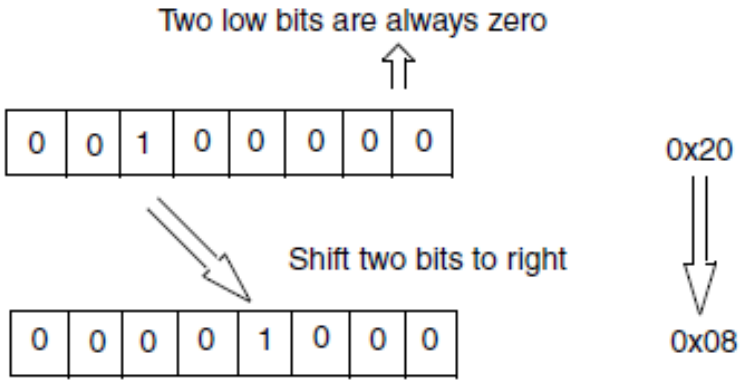
USB RAM: adresa bufferu EP

- Položka EPADR v buffer deskriptoru se vypočte jako relativní adresa počátku bufferu EP v USB RAM posunutá o 2 bity doprava
- Příklad: Na EP0 vystavíme 8 bytů dlouhý paket, který bude následně odeslán počítači. Nechť buffer EP0 začíná na adrese 0x1880. Jak bude vypadat deskriptor bufferu?

Registers	Contents
Status & Control Reg	0x00
BC[7:0]	0x08
EPADR[9:2]	0x08

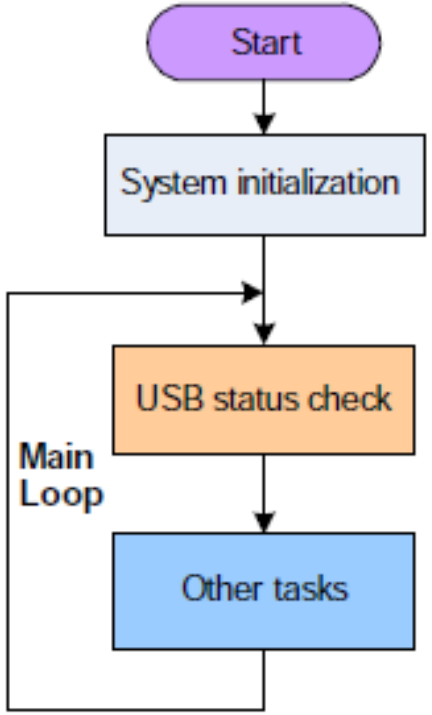
Výpočet EPADR

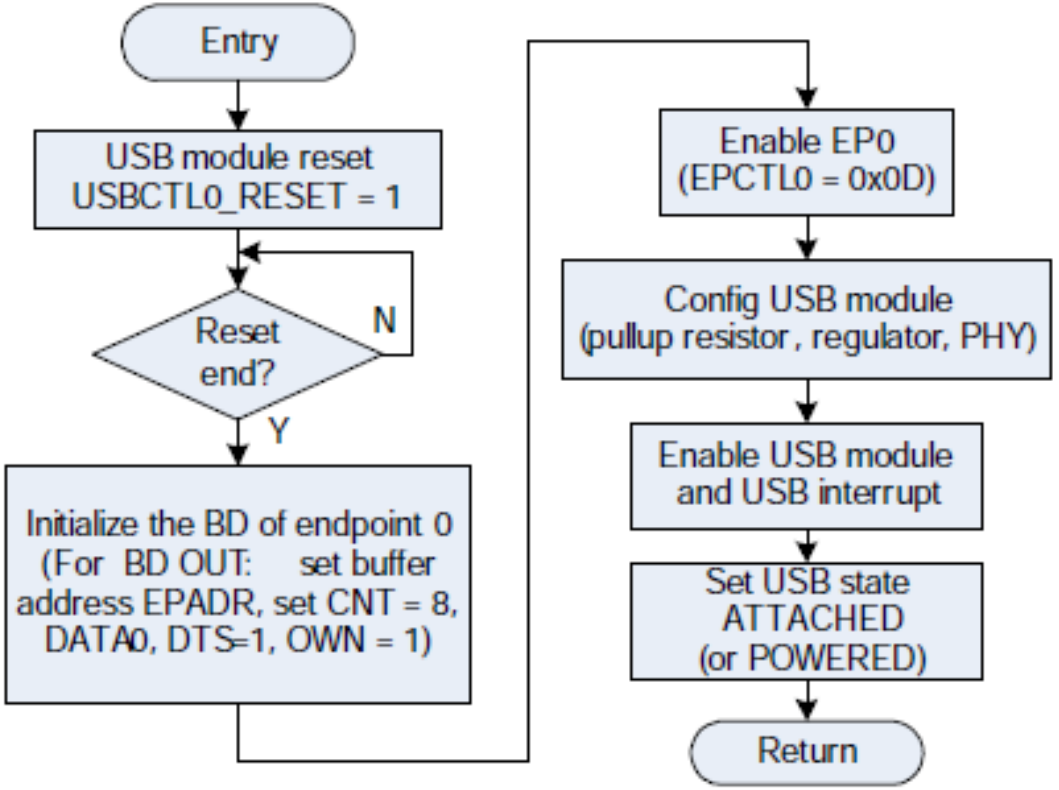




Struktura aplikace (firmware zařízení)

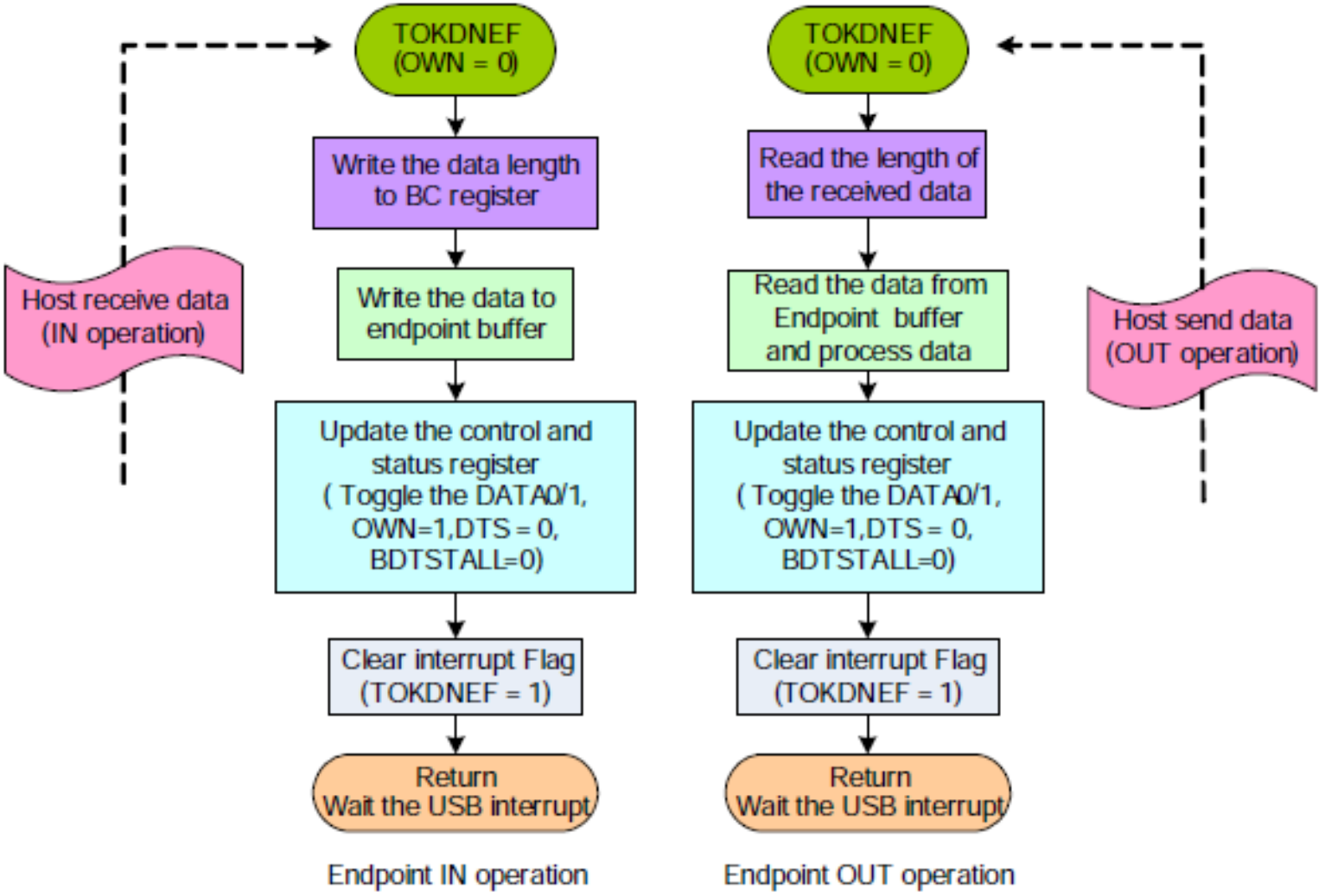
- Typický obslužný program USB zařízení má následující strukturu. Pro detekci stavu USB modulu je možné využít dotazovací smyčku, případně přerušení.





Komunikace na endpointech USB zařízení

- Princip spočívá v nastavení deskriptorů bufferů EP a předání řízení systému SIE, který zajistí přenos a nastavení příslušných příznaků.

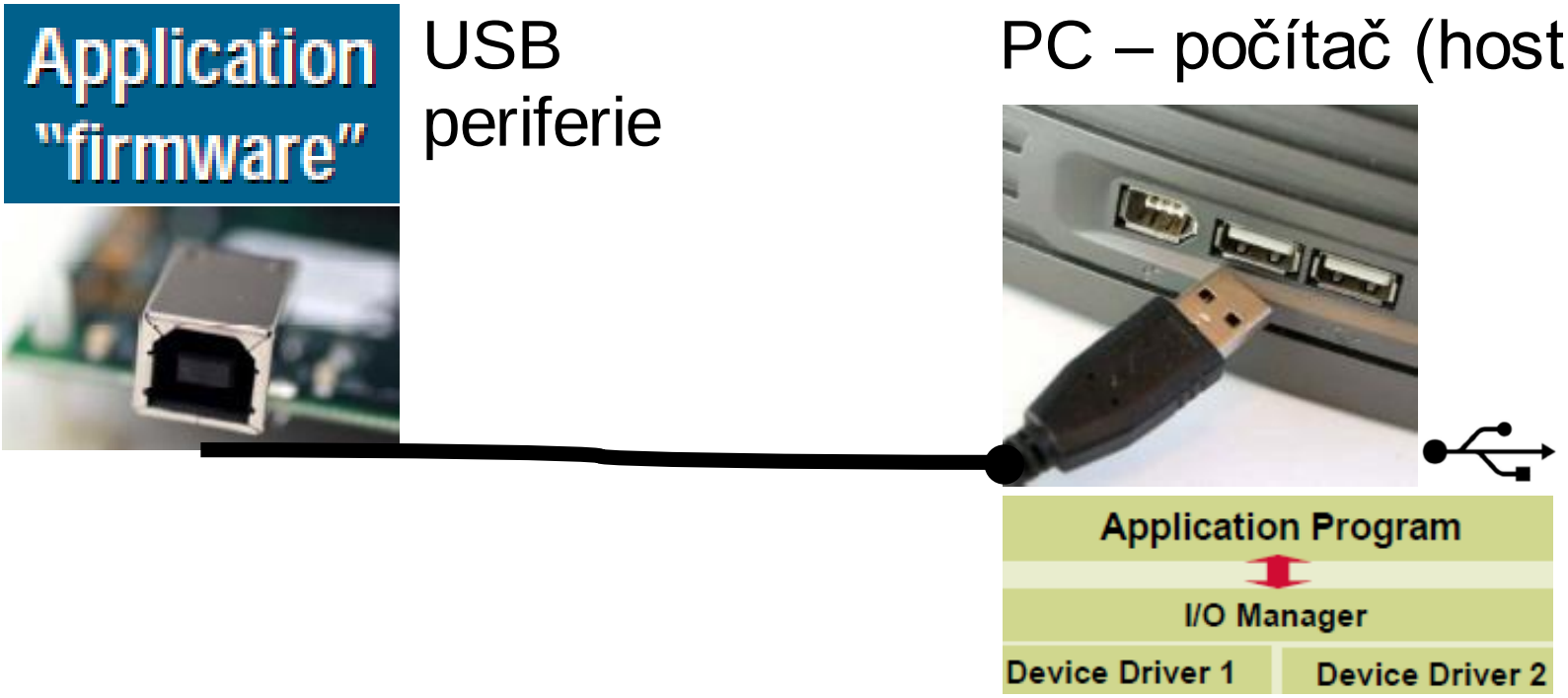


Část 3: návrh firmwaru pro MCU – vlastní programátorská činnost

Požadavky při návrhu USB zařízení

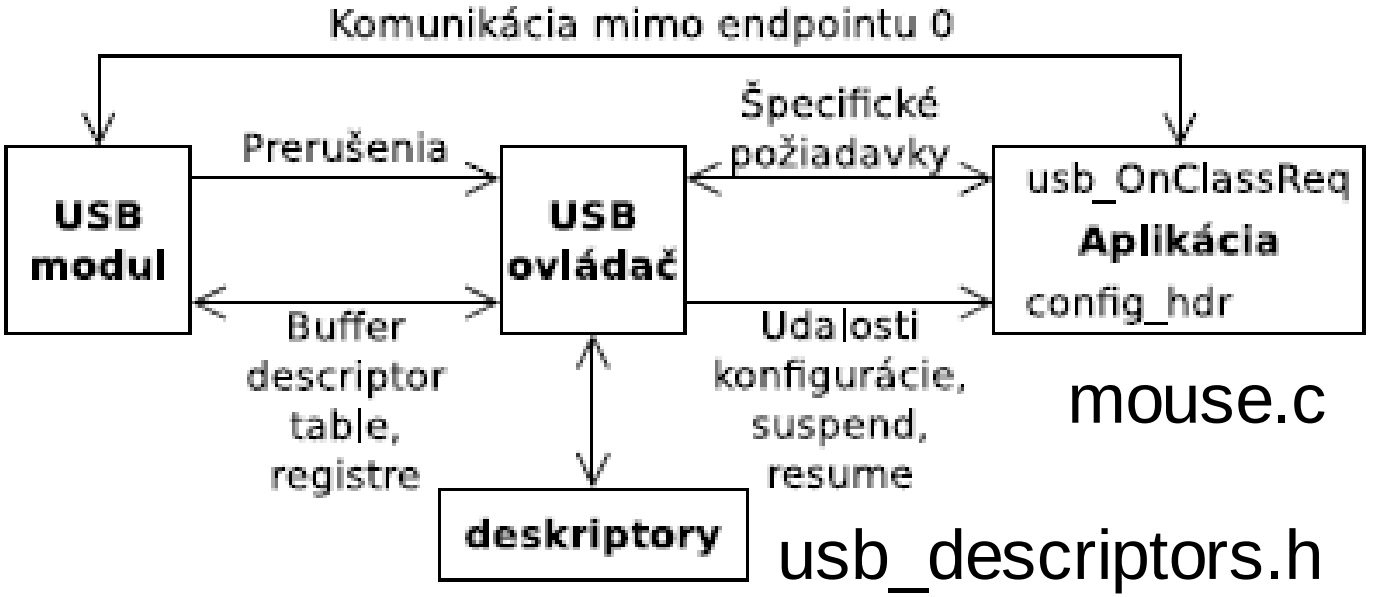
součást: potřebná komponenta (máme/bude vytvořena)

- HW: systém vybavený USB modulem (MC9S08JM60)
- SW: ovladač na straně počítače (standardní HID mouse)
- FW: obslužný program pro myš (viz příklad dále)



Příklad: USB myš – popis implementace

- Základní moduly aplikace
  - USB modul na JM60 (HW)
  - Ovladač USB modulu (usb_drv.c, usb_ep0.c)
  - Popis funkce aplikace (mouse.c)



mouse.c

usb_descriptors.h

Deklarace buffer deskriptoru v C

```
//endpoint descriptor
struct usb_ep_descr {
        union {
                struct {
                        unsigned :1;
                        unsigned :1;
                        unsigned BDTSTALL:1;/* TKPID */
                        unsigned DTS:1; /* TKPID */
                        unsigned :1; /* TKPID */
                        unsigned :1; /* TKPID */
                        unsigned DATA:1;
                        unsigned OWN:1;
                }bits;
                uint8_t reg;
        }u;
        uint8_t bc; /* byte count */
        uint8_t epadr; /* adresa endpointu v USB RAM */
};
```

Deklarace buffer deskriptoru table

```
//buffer descriptor table
struct usb_bdt { /* 32B */
        struct usb_ep_descr ep0in;
        struct usb_ep_descr ep0out;
        struct usb_ep_descr ep1;
        struct usb_ep_descr ep2;
        struct usb_ep_descr ep3;
        struct usb_ep_descr ep4;
        struct usb_ep_descr ep5even;
        struct usb_ep_descr ep5odd;
        struct usb_ep_descr ep6even;
        struct usb_ep_descr ep6odd;
        uint8_t _pad[2]; /* reserved */
};
```

Deklarace struktury SETUP paketu

```
//format setup transakcie a hodnoty jej poloziek
struct usb_setup_pkt {
        uint8_t bmRequestType;
        uint8_t bRequest;
        uint16_t wValue;
        uint16_t wIndex;
        uint16_t wLength;
};

struct usb_setup_pkt_bytes { //viacbajtove polozky pristupne po bajtoch
        uint8_t bmRequestType;
        uint8_t bRequest;
        uint8_t wValue[2];
        uint8_t wIndex[2];
        uint8_t wLength[2];
};

union ep0out_u {
        uint8_t buf[USB_EP0_BUF_SIZE];
        struct usb_setup_pkt pkt;
        struct usb_setup_pkt_bytes bytes;
        struct usb_setup_pkt_std {
                uint8_t bmRequestType;
                uint8_t bRequest;
                uint16_t wValue;
                uint16_t wIndex;
                uint16_t wLength;
        }std;
```

Proměnné struktur BDT a EP0 bufferů: nutno dodržet umístění (dáno výrobcem JM60)

```
//buffre endpointu 0
struct usb_bdt bdt                          @0x1860;
union ep0out_u ep0out_setup                 @0x1880;
unsigned char ep0in_buf[USB_EP0_BUF_SIZE]   @0x1890;
```

!

Funkce aplikace

- SW realizuje inicializaci MCU a USB modulu
- V rámci hlavní smyčky přejde MCU do wait-režimu (povolí přerušení, zastaví CPU, čeká na přerušení)
- Vše ostatní je realizováno jako obsluha přerušení

```
void main(void) {
        //write once registre
        SOPT1 = 0b00110000;  //COP vypnuty, povoli stop3 rezim
        SPMSC1 = 0b00010100; //detekcia podpetia a reset mimo stop3 rezimu
        SPMSC2 = 0b00000000; //2.74V<=  ^, stop3 rezim

        usb_clock_init();    //12MHz krystal, bus 24MHz, jadro 48MHz => USB 48MHz
        usb_init();          //zapne USB modul a inicializuje USB ovladac

        EnableInterrupts;
        for(;;) {
                asm wait;
        }
}
```

main.c

Činnost aplikace po připojení k PC

- Počítač generuje USB bus reset, na základě čehož je třeba uvést USB modul JM60 do určitého definovaného počátečního stavu

```
__interrupt VectorNumber_Vusb void usb_isr(void) {
        .
        .
        .

        if(INTSTAT_USBRSTF && INTENB_USBRST) {
                //USB bus reset signal
                usb_OnReset();
                INTSTAT_USBRSTF = 1;
        }

        .
        .
        .
}
```

usb_drv.c

Funkce usb_OnReset

- usb_OnReset mimo jiné provádí
  - Nastavení adresy USB zařízení

    ADDR = 0;
  - Nastavení adres bufferů EP0 IN, EP0 OUT

    bdt.ep0in.epadr = EPADR(ep0in_buf);
    bdt.ep0out.epadr = EPADR(ep0out_setup);
  - Nastavení EP0 – EP7

    EPCTL0 = EP_RX | EP_TX | EP_HSK;
    EPCTLn = 0;
  - Nastavení stavu USB zařízení

    usb_state = USB_DEFAULT;

usb_drv.c

Komunikace s hostitelským PC

- Zpracování paketu je signalizováno nastavením příznaku TOKDNEF v registru INTSTAT

usb_drv.c

```
__interrupt VectorNumber_Vusb void usb_isr(void) {
        .
        .
        .
        if(INTSTAT_TOKDNEF && INTENB_TOKDNE) {
                //dokoncena transakcia
                uint8_t ep = (STAT & 0xF0) >> 4;
                if(ep == 0)
                        //obsluha endpointu 0 je radsej mimo obsluznych funckii
                        ep0_hdr();
                else
                        //zavola obsluznu funkciu endpointu v aplikacii
                        ep_hdrs[ep]();
                INTSTAT_TOKDNEF = 1;
        }
        .
        .
        .
}
```

registr STAT

	7	6	5	4	3	2	1	0
R	ENDP[3:0]				IN	ODD	0	0
W								

Funkce ep0_hdr (EP0 Handler)

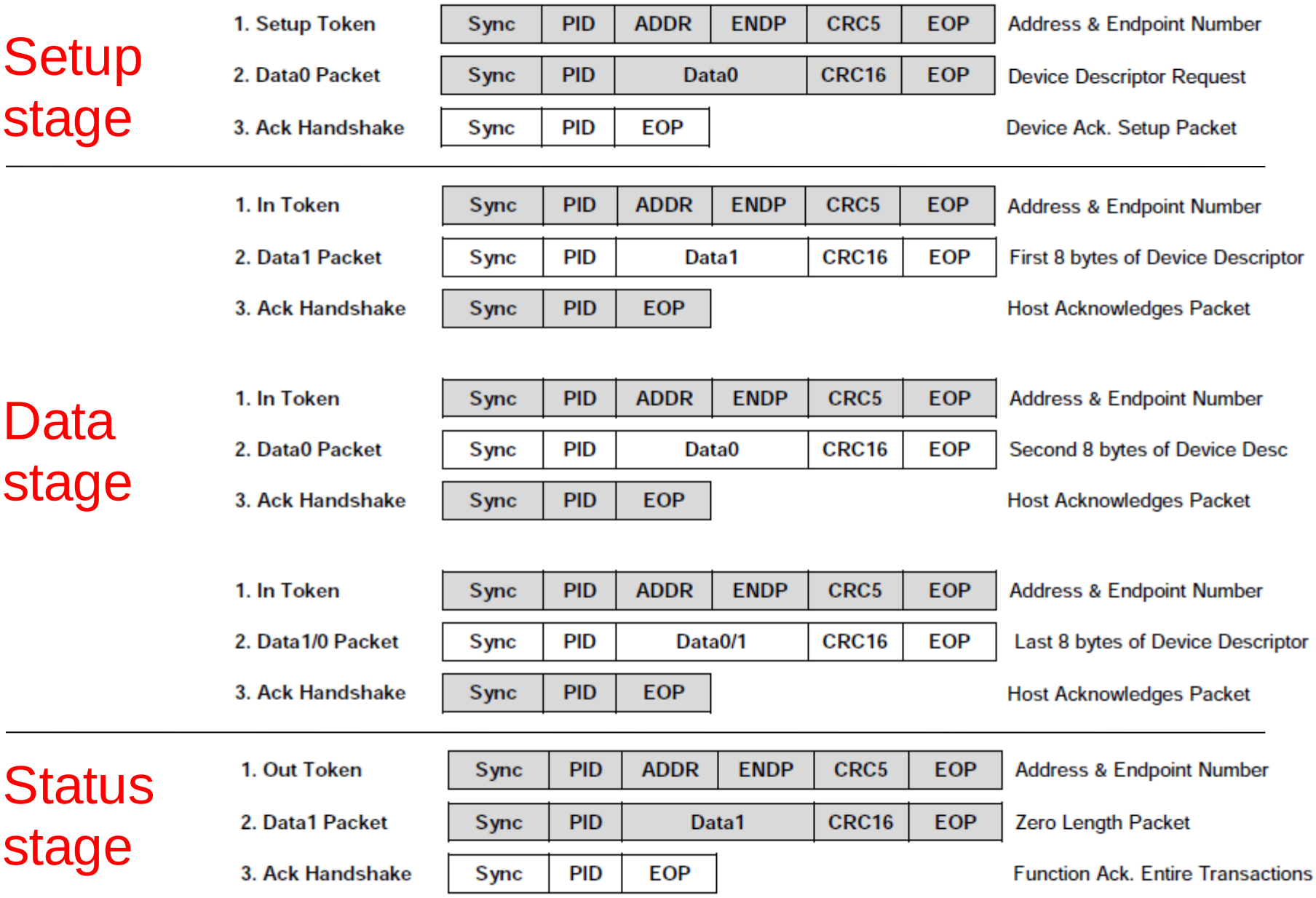
- Tato funkce zjistí směr přenosu (IN/OUT) z registru STAT a podle toho realizuje další činnost.
- Například: při enumeraci byl přijat (OUT, PC-->USB zařízení) SETUP paket, z něhož je třeba vyčíst typ požadavku a ten pak příslušnou funkcí obsloužit.

```
static void ep0_setup(void) {
        .
        .
        .
        uint8_t req = ep0out_setup.bytes.bRequest;

        if(usb_state >= USB_DEFAULT) {
                /* povolene prikazy: GET_DESCRIPTOR, SET_ADDRESS (mimo
                   CONFIGURED stavu)*/
                if(req == GET_DESCRIPTOR) {
                        ok = get_descriptor_hdr();
                } else if(req == SET_ADDRESS) {
                        ....
                }
        }
```

usb_ep0.c

Přenos na EP0 – příklad zpracování žádosti GET_DESCRIPTOR od PC



Realizace přenosu dat

- Obecně probíhá vyřizování požadavků tak, že MCU vloží data do bufferu příslušného EP, nastaví deskriptory v BDT, případně registry EPCTLn, a předá řízení modulu SIE:

```
static uint8_t ep0_tx_data(uint8_t *data, uint16_t bytes) {
        //vyberie tolko dat kolko sa zmesti do buffra endpointu
        uint8_t limit;
        if(bytes > sizeof ep0in_buf)
                limit = sizeof ep0in_buf;
        else
                limit = (uint8_t)bytes;

        //vlozi data do buffra endpointu ak tam este niesu (NULL)
        if(data != NULL) {
                uint8_t pos;
                for(pos = 0; pos < limit; pos++, data++)
                        ep0in_buf[pos] = *data;
        }

        //posle data (nastavenim SIE=1 v reg. BDT)
        bdt.ep0in.bc = limit;
        if(bdt.ep0in.u.bits.DATA)
                bdt.ep0in.u.reg = EPF_OWN_SIE | EPF_DTS | EPF_DATA1;
        else
                bdt.ep0in.u.reg = EPF_OWN_SIE | EPF_DTS | EPF_DATA0;

        return limit;
}
```

Realizace funkce zařízení

- SW na MCU musí zajistit korektní průběh procesu enumerace a dále též implementovat vlastní funkci USB zařízení.
- Veškerá komunikace s PC probíhá v podobném duchu, jak bylo naznačeno na předchozích slidech.
- V případě myši je nutné implementovat:
  - způsob pohybu kurzoru,
  - tlačítka myši, případně kolečko,
  - přenos těchto dat do PC (zde na endpointu č. 1).

Realizace funkce zařízení

- Požadavky od PC, které souvisí s obsluhou zařízení třídy HID (tzv. class-specific requests) jsou řešeny v rámci modulu aplikace.

usb_ep0.c

```
static void ep0_setup(void) {
        .
        .
        .
        if(ep0out_setup.bmRequestType.Type != TYPE_STD)
                //specificke prikazy (class-specific requests)
                ok = usb_OnClassReq();
        else {
                .
                .
                .
```

→ mouse.c

  - funkce usb_OnClassReq – zpracování class-specific požadavků (např. zaslání deskriptoru HID, info o rozhraní, nastavení konfigurace apod.).

Realizace funkce zařízení

- Přenosy související s realizací funkce myši (pohyby kurzoru, stav tlačítek atd.) probíhají na EP1 a jsou taktéž řešeny v modulu aplikace. Postup je obdobný (data do EP, nastavení v BDT, předání řízení SIE).

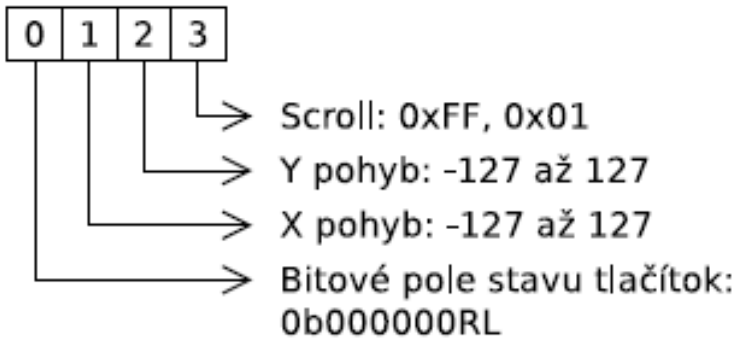
usb_drv.c

```
__interrupt VectorNumber_Vusb void usb_isr(void) {
        .
        .
        .
        if(INTSTAT_TOKDNEF && INTENB_TOKDNE) {
                //dokoncena transakcia
                uint8_t ep = (STAT & 0xF0) >> 4;
                if(ep == 0)
                        //obsluha endpointu 0 je radsej mimo obsluznych funckii
                        ep0_hdr();
                else
                        //zavola obsluznu funkciu endpointu v aplikacii
                        ep_hdrs[ep]();
                INTSTAT_TOKDNEF = 1;
        }
        .
        .
        .
}
```

→ mouse.c

Realizace funkce zařízení

- Funkce myši není vázána na žádné konkrétní zařízení, je tedy možné její chování vhodně emulovat, například
  - kolečko (scroll-up, scroll-down) může být nahrazeno dvojicí tlačítek, která po stisku nastaví příslušný příznak, jenž je počítačem interpretován jako scroll,
  - pohyb kurzoru pomocí směrových tlačítek, joysticku, akcelerometru apod.
- Realizace dle specifikace HID:
  - scroll generuje definované hodnoty (příkazy),
  - Data pro pohyb kurzoru jsou dána diferencí vůči předchozí poloze
  - Binární příznaky tlačítek



Literatura a zdroje pro další studium

- M. Malý: USB 2.0, díly 1, 2, 3. Redakce HW serveru, 2005, dostupný na http://hw.cz/Rozhrani/ART1244-USB-2.0---dil-2.html
- J. Axelson: USB Complete, third edition. Lakeview Research, 2005
- C. Peacock: USB in a Nutshell, third release, 2002, dostupný na http://www.beyondlogic.org/usbnutshell/usb-in-a-nutshell.pdf
- G. Kroah-Hartman: Writing a Simple USB Driver. Linux Journal, issue #120, April 2004, dostupný na http://www.linuxjournal.com/article/7353
- G. Kroah-Hartman: How to Write a Linux USB Device Driver. Linux Journal, issue #90, October 2001, dostupný nahttp://www.linuxjournal.com/article/4786
- D. Liu: USB Device Development with the MC9S08JM60, Application Note AN3560. Freescale Semiconductor, 2008, dostupný na http://cache.freescale.com/files/microcontrollers/doc/app_note/AN3560.pdf
- AN3560 Software – HID Mouse, Freescale Semiconductor, dostupný přes http://www.freescale.com/webapp/sps/download/license.jsp?colCode=AN3560SW&location=null&fpsp=1
- USB Implementer's Forum: Device Class Definition for Human Interface Devices, ver. 1.11, 2001, dostupný na http://www.usb.org/developers/devclass_docs/HID1_11.pdf
- Universal serial Bus Specification, rev.2.0, 2000, dostupný na http://www.usb.org/developers/docs/usb_20_081810.zip
- MQP Electronics Ltd.: USB Made Simple, 2008, dostupný na http://www.usbmadesimple.co.uk/
- M. Riša: Návod a praktické příklady k tvorbě jednoduchých vestavných USB aplikací založených na MC9S08JM60 [bakalářská práce]. FIT VUT v Brně, 2012