VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

USB KEYLOGGER

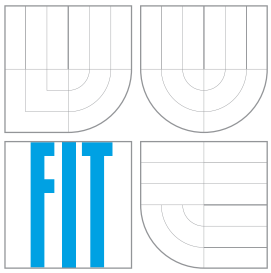
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB LOJDA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

USB KEYLOGGER

USB KEYLOGGER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB LOJDA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ZDENĚK VAŠÍČEK, Ph.D.

BRNO 2013

Abstrakt

Práce se zabývá návrhem a implementací USB zařízení pro záznam stisků kláves. Úvodní část je věnována teoretickým poznatkům, nezbytným pro porozumění použitých technologií. Praktická část se věnuje zejména využití dostupných ovladačů USB HID a implementací vlastního ovladače USB Mass-storage v režimu slave. Následuje část o realizaci zařízení za pomocí obvodu Vinculum II včetně návrhu DPS.

Abstract

This thesis describes design and implementation of USB keylogger. The introductory section discusses the teoretical knowledge necesarry for understanding the practical part. Practical part pays attention to available drivers, USB HID and implementation of a custom driver for USB Mass-storage in USB slave mode. Finally there is a section on implementation of the device using Vinculum II MCU, including PCB design.

Klíčová slova

USB keylogger, USB, HID, BOMS, Mass Storage, MCU, firmware, ovladač USB, vestavěný systém

Keywords

USB keylogger, USB, HID, BOMS, Mass Storage, MCU, firmware, USB driver, embedded system

Citace

Jakub Lojda: USB keylogger, bakalářská práce, Brno, FIT VUT v Brně, 2013

USB keylogger

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Zdeňka Vašíčka, Ph.D.

.....
Jakub Lojda
14. května 2013

Poděkování

Tímto bych rád poděkoval vedoucímu práce, panu Ing. Zdeňku Vašíčkovi, Ph.D., za odbornou pomoc při řešení odhalených problémů, pomoc při výrobě DPS, škole za financování výroby DPS a vývojářům firmy FTDI za kvalitní podporu dodávaných produktů.

Obsah

Kapitola 1

Úvod

Keylogger je prostředek, který umožňuje zaznamenávání stisků kláves počítače pro pozdější analýzu. Vedle SW keyloggerů, které je možné odhalit antivirovým programem, existují HW keyloggery, které je zpravidla těžší odhalit. Jde o zařízení, které se připojí mezi klávesnici a počítač. Záznam stisknutých kláves probíhá do vnitřní paměti, která je zpravidla zpřístupněna po stisku určité kombinace kláves.

USB keylogger je HW keylogger, který zaznamenává stisknuté klávesy na připojené USB klávesnici. Tato práce se zabývá uvedením do dané problematiky, návrhem a implementací takového zařízení v praxi.

Úvodní teoretická část práce se věnuje popisu sběrnice USB v kapitole 2, SD kartám v kapitole 3 a zvolenému MCU Vinculum VNC2 v 4. Kapitola 5 pojednává o návrhu a realizaci zařízení. Následuje popis praktické realizace v kapitole 6.

1.1 Cíle práce

Cílem práce je navrhnout a prakticky realizovat HW zařízení, které by bylo možné zapojit mezi USB klávesnici standardu HID a osobní počítač. Zařízení by mělo zaznamenávat klávesy stisknuté na připojené klávesnici a ukládat je na vloženou microSD kartu.

Veškerá data uložená na kartě by měla být šifrována tak, aby nebylo možné je vyčíst prostým vložením karty do čtečky. Zařízení by mělo reagovat na určitou kombinaci kláves zpřístupněním tohoto úložiště.

Zařízení by mělo být možné využívat i pro ukládání dat na kartu, tzn. zpřístupněné úložiště po stisku určité kombinace kláves musí být rovněž zapisovatelné ze strany počítače.

Mým osobním cílem bylo seznámení se s návrhem aplikací pro USB a zaměřit se na získání zkušeností s obvodem Vinculum VNC2 firmy FTDI tak, aby bylo možné je později využít při tvorbě jiných USB aplikací pro tuto platformu.

Kapitola 2

Sběrnice USB

Kapitola čerpá ze zdrojů [11], [14], [20] a [23].

USB je společným dílem sedmi firem, které v roce 1994 započaly vývoj. Byly to Compaq, DEC, IBM, Intel, Microsoft, NEC a Nortel. Jejich cílem bylo implementovat rozhraní, které by umožnilo jednoduché připojení periferií k počítači bez nutnosti restartovat operační systém, nebo dokonce počítač po dobu montáže úplně vypnout. USB je tady zaměřeno hlavně na jednoduchost použití ze strany uživatele.

První oficiálně publikovanou verzí bylo USB 1.0 v roce 1996, která byla o dva roky později nahrazena revizí 1.1. USB 1.x specifikuje dva druhy přenosů, jsou to tzv. *Low-Speed* přenosy s až 1,5 Mb/s a *Full-Speed* s rychlostí až 12 Mb/s. V roce 2001 došlo ke schválení USB 2.0, které umožňovalo přenosovou rychlost až 480 Mbit/s označované jako režim *High-Speed*.

Následující verze USB 3.0 pochází z roku 2008 a je charakteristická zavedením nových datových vodičů a tudíž zvýšenou propustností až na 5 Gb/s v režimu *Super-Speed*. Specifikuje navíc mj. režim nízké spotřeby a je zpětně kompatibilní s USB 2.0.

2.1 Charakteristika sběrnice USB

USB je univerzální sériové rozhraní, využívané pro připojení periferií nejrůznějších typů. Jedná se o řízenou sběrnici, tzn. veškeré činnosti zahajuje hostitelský řadič.

Na sběrnici se vyskytuje pouze jedno zařízení typu master. Toto zařízení je provozováno v tzv. režimu USB Host, jde o kořenový rozbočovač. Přes tento rozbočovač komunikuje operační systém s připojenými zařízeními. Komunikace probíhá výhradně metodou vyzývání, tzv. polling. Tím je zajištěn výlučný přístup ke sběrnici.

Připojované periferie lze rozdělit do dvou skupin. Na zařízení rozšiřující sběrnici, která neposkytují žádnou funkčnost, pouze přidávají nové USB porty. Jde o rozbočovače, které zaberou jeden port ale více jich vytvoří. Druhou skupinou jsou zařízení poskytující nějakou konkrétní funkčnost, jedná se např. o zařízení typu flash disk, klávesnice apod.

Sběrnice tedy tvoří hierarchii vzájemně propojených rozbočovačů a funkčních zařízení. Dle specifikace smí tato hierarchie mít maximálně sedm úrovní. Maximální počet současně připojených zařízení je 127.

2.2 Přenosy v rámci sběrnice USB

Celý systém lze chápat jako množinu logických propojení jednotlivých endpointů zařízení a aplikací (ovladačů) pro daná zařízení, tzv. rour. Endpoint je vnitřním prvkem zařízení, do něhož se informace zapisují a ze kterého se informace čtou.

Podle [19] jsou definovány následující přenosové rychlosti:

Low-speed: definováno v USB 1.0, přenosová rychlost do 1,5 Mbit/s, primárně určeno pro zařízení typu HID (Human Interface Device), která jsou obecně méně datově náročná.

Full-speed: definováno v USB 1.0, přenosová rychlost do 12 Mbit/s, výchozí rychlost USB 1.0, rozbočovače musí podporovat minimálně tuto rychlost. Principiálně jde o podobnou komunikaci jako u low-speed, s rozdílem vyšší rychlosti.

High-speed: definováno v USB 2.0, přenosová rychlost do 480 Mbit/s, všechna zařízení této specifikace musí zpětně podporovat také režimy USB 1.x.

Super-speed: definováno v USB 3.0, přenosová rychlost 5 Gbit/s, přidává nové vodiče, možnost paralelní komunikace přes více rour, zůstává však zpětně kompatibilní.

Podrobnosti jsou dohledatelné v [24].

2.3 Principy komunikace ve sběrnici USB

Komunikace ve sběrnici USB je rozložena na tzv. pakety. Vrámci těch probíhá výměna vícebytových hodnot podle [32] v konvenci little-endian, tzn. nejméně významné byty jsou odeslány nejdříve.

Každý rámec na začátku obsahuje tzv. PID, Packet Identifier. Ten je čtyřbitový, tedy lze rozlišit celkem 16 druhů paketů ve sběrnici USB. Čtyři bity PID na začátku rámce jsou zdvojeny, přičemž druhá posloupnost čtyř bitů je bitově invertovaná. Tato skutečnost zajišťuje určitou schopnost detekovat chyby při přenosech.

Obecně lze rozlišit tři typy paketů, *řídící (tzv. token)* pakety, *datové* pakety a *handshake* pakety.

2.3.1 Řídící (token) pakety

Řídící pakety, nazývané též token pakety, se skládají z úvodního bytu PID, následují data, 11 bitů adresy. Tuto adresu lze rozložit na 7 bitů adresu zařízení a 4 bity adresu endpointu daného zařízení. Z délky adresy plyne omezení pouze 127 zařízení na jedné sběrnici. Adresa zařízení "0", stejně jako adresa endpointu "0", je implicitní a využívají ji právě připojená zařízení k inicializaci a nastavení zařízení. Za adresou následuje pět bitů CRC (Cyclic Redundancy Check) kontrolního součtu.

Sync	PID	ADDR	ENDP	CRC5	EOP

Obrázek 2.1: Formát řídícího (tzv. token) paketu, přejato z [28]

Lze rozlišit celkem šest typů token paketů. Pakety typu IN a OUT slouží k adresaci koncového zařízení. Pokud koncové zařízení obdrží paket typu IN se svou adresou, může kořenovému řadiči odpovědět paketem typu STALL, kdy dává najevo svůj chybový stav,

paketem typu NAK, pokud zařízení detekovalo chybu při přenosu. Pokud nenastala chyba, obdrží kořenový řadič paket typu DATAx s vyžádanými daty. V takovém případě čeká zařízení na přijetí token paketu ACK od kořenového řadiče.

Pakety OUT slouží k odeslání dat směrem k zařízení. Po tomto paketu následuje datový paket. Zařízení odpovídá ACK, pokud byl přenos úspěšně dokončen, NAK, NYET příp. STALL, pokud došlo k chybě.

Paket SOF, Start of frame, slouží k synchronizaci isochronních datových přenosů. Jde o paket vysílaný každou milisekundu kořenovým řadičem, který má na místě adresy 11 bitové číslo rámce, zařízení v režimu USB 2.0 dostávají tzv. mikrorámce každých 125 μs.

SETUP token pakety slouží k inicializaci právě připojeného zařízení.

V USB 2.0 byly navíc zavedeny token pakety typu SPLIT a PING. Token pakety typu SPLIT se užívají při realizaci komunikace rychlostí high-speed, i když ne všechna zařízení na sběrnici tuto rychlost podporují. Tím je zajištěno optimální využití a pomalá zařízení tak nebrzdí rychlé přenosy.

2.3.2 Datové pakety

Datový paket začíná idenfikikátorem PID, následuje 0 - 8192 bitů samotných dat, v režimu low-speed je zde však maximálně pouze 16 bitů dat, nakonec je zde obsaženo pět bitů CRC.

Datovým paketům předchází token paket typu IN, který obsahje adresu, tzn. že datové pakety již neobsahují adresu příjemce, neboť komunikace je již navázána.

Datové pakety lze dále rozlišit na pakety typu DATA0 a DATA1, od verze USB 2.0 také DATA2 a MDATA. Typem DATA0 jsou označovány sudé a typem DATA1 liché pakety. Toto rozlišení je nutné pro implementaci základní spolehlivé komunikace, nazývané stop-and-wait, která je blíže popsána v [21].

Způsob komunikace stop-and-wait zajišťuje spolehlivé doručení paketů a také zachování pořadí. Jedná se o mechanizmus detekce a případné opravy chyby při snaze komunikovat spolehlivě po nespolehlivé lince. K funkčnosti využívá koncepci upozornění o úspěšném přijetí dat (acknowledgement, ACK) a časovače (timeout). Zpočátku je nastaven časovač, následně jsou odeslána data. Příjemce po obdržení dat odpoví paketem ACK. Pokud odesílatel nedostane toto upozornění, data nebo ACK paket mohl být ztracen. Po vypršení časovače tedy odesílatel odesílá datový paket znovu. Pokud příjemce obdrží dvakrát stejný paket, mohlo dojít ke ztrátě paketu ACK, odešle tedy upozornění ACK, ale přijatá data ignoruje. Kvůli této funkčnosti je nutné, aby si obě komunikující zařízení udržovala jednobitový stav, určující, zda poslední odeslaný paket byl sudý (DATA0) nebo lichý (DATA1). Rozsah jeden bit plyne z principu funkce stop-and-wait, kdy je najednou odeslán vždy pouze jeden paket.

Pakety typu DATA2 a MDATA se používají od verze USB 2.0 pro vysokorychlostní isochronní přenosy. Další informace o datových paketech je možné dohledat v [20].

Sync	PID	DATA	CRC16	EOP

Obrázek 2.2: Formát datového paketu, přejato z [28]

2.3.3 Handshake pakety

Také handshake pakety začínají identifikátorem PID. Podle tohoto identifikátoru můžeme rozlišit celkem čtyři typy těchto paketů. Jedná se pakety typu ACK, které potvrzují přijatá

data ze strany příjemce, pakety typu NAK, který naopak žádá o znovuzaslání dat, např. kvůli chybě, která nastala při přenosu a detekci takové chyby pomocí CRC a od USB verze 2.0 zavedené pakety typu NYET a STALL.

Při výzvě o data od kořenového řadiče může nastat stav, kdy koncové zařízení ještě není připraveno k dalšímu přenosu, např. proto, že má zaplněny přijímací buffery. K informování kořenového řadiče o této skutečnosti slouží paket typu NYET. Pokud kořenový řadič obdrží při dotazování zařízení tuto odpověď, začne se periodicky dotazovat koncového zařízení token pakety typu PING, na které zařízení neodpovídá paketem ACK, dokud je stále zaneprázdněno. Jakmile kořenový řadič dostane od takového zařízení paket ACK, pokračuje v komunikaci s tímto zařízením tam, kde skončil.

Paket typu STALL je odeslán, pokud se zařízení nachází v chybovém stavu a je třeba provést nápravu.

Sync	PID	EOP

Obrázek 2.3: Formát handshake paketu, přejato z [28]

2.4 USB deskriptory

Každé USB zařízení obsahuje hierarchii deskriptorů popisujících samotné zařízení, výrobce, podporovanou verzi USB apod. Podle [28] můžeme deskriptory rozdělit do skupin podle toho, co popisují, na tzv.:

- Device deskriptory
- Configuration deskriptory
- Interface deskriptory
- Endpoint deskriptory
- String deskriptory

USB zařízení má pouze jeden Device deskriptor, který popisuje verzi USB, Product ID, Vendor ID apod. Zařízení může mít více Configuration deskriptorů, ze kterých USB host během enumerace vybírá. V jednu chvíli může být vybrána jedna konfigurace. Interface deskriptory lze chápat jako logické spojení Endpoint deskriptorů, které k sobě patří a zajišťují určitou funkčnost. Endpoint deskriptory se používají pro popis endpointů, kromě nultých. USB Host používá informace z těchto deskriptorů pro odvození šířky přenosového pásma. String deskriptory jsou určeny pro textový popis zařízení v kódování unicode. Pokud nejsou použity, je nastavena nulová délka.

2.5 Rozhraní USB

USB kabely sestávají z tzv. kroucených dvojlinek. Tento způsob vedení vodičů slouží k eliminaci přeslechů a šumu.

Každý USB 1.x a 2.0 kabel obsahuje celkem čtyři vodiče, tj. dva kroucené páry. Přehled vodičů a příslušnost k pinům konektrů je znázorněn v tabulce 2.1.

Pin	Název signálu	Popis
1	Vcc	Napětí +5V
2	Data -	Záporný datový vodič
3	Data +	Kladný datový vodič
4	GND	Zem

Tabulka 2.1: Popis vodičů v kabelech USB 1.x a USB 2.0, přejato z [20]

Maximální délka kabelu závisí na použité rychlosti přenosu. U přenosů Low-speed je to 5 metrů, přenosy Full-speed by měly fungovat do délky kabelu 12 metrů. Přenosy USB 2.0 High-speed jsou limitovány délkou kabelu 5 metrů, tento fakt je způsoben maximální možnou prodlevou při komunikaci.

V technologii USB 3.0 je maximální přenosová rychlost dosažena zvýšením počtu vodičů na celkový počet osm vodičů. Zařízení tak mohou komunikovat více rourami zároveň. Přehled značení vodičů je v tabulce 2.2.

Pin	Název signálu	Popis
1	Vcc	Napětí +5V
2	Data -	Záporný diferenční datový vodič
3	Data +	Kladný diferenční datový vodič
4	GND	Zem
5	SSRX-	Super-speed přijímací diferenční pár
6	SSRX+	
7	GND DRAIN	Zem pro signály
8	SSTX-	Super-speed vysílací diferenční pár
9	SSTX+	

Tabulka 2.2: Popis vodičů v kabelech USB 3.0, přejato z [22]

Maxilání použitelná délka kabelů USB 3.0 není normou specifikována, ovšem v praxi je použitelná délka přibližně 3 metry.

2.6 Třídy USB zařízení

Následující část čerpá z [1].

Zařízení schopná komunikovat po sběrnici USB je možné rozdělit do různých tříd. Dělení probíhá na základě podobnosti činností zařízení.

Důležité třídy z pohledu implementace USB keyloggeru jsou Human Interface (HID) a BOMS (Bulk-only Mass Storage), kterými se bude zabývat následující popis.

2.6.1 Zařízení třídy HID

Třída HID USB zařízení zahrnuje periferie, které umožňují hostujícímu zařízení reagovat na okolní podněty. Jedná se mj. o klávesnice. Reakce by měla být co nejrychlejší, aby nedocházelo k nepříjemným prodlevám od provedení podnětu do provedení přiřazené akce. To je mj. důvod, proč informace o podnětech z okolního světa putují po sběrnici zakódované

do poměrně malých paketů. Tyto se nazývají report pakety. Jejich formát je dán report deskriptorem, typickým pro třídu HID. Z výše uvedeného tedy plyne, že spíše než rychlost komunikace je nutná okamžitá odezva.

Kromě výše zmíněného report protokolu existuje také boot protokol, který je jeho jednodušší variantou. Nevyužívá report deskriptoru. Jeho formát je pevně dán.

Podle specifikace HID [18] je pro HID zařízení vymezeno celkem šest třídně-specifických příkazů. Jedná se o následující:

GET_REPORT Umožňuje hostiteli přijmout report data pomocí ovládací roury

SET_REPORT Umožňuje hostiteli odeslat report data zařízení

GET_IDLE Slouží k přečtení momentální doby spánku zařízení

SET_IDLE Pozastaví zasílání report paketů dokud nevyprší specifikovaná doba, nebo dokud nedojde k události

GET_PROTOCOL Zjistí, který protokol je momentálně aktivní (boot/report), podporován zařízeními v podtřídě Boot.

SET_PROTOCOL Přepíná mezi boot/report protokolem

2.6.2 Zařízení třídy BOMS

Třída BOMS zahrnuje zařízení přenášející soubory v jednom nebo obou směrech. Jedná se např. o CD-ROM mechaniky, floppy mechaniky, USB flash disky apod. Každé zařízení dodržuje určitý standard příkazových bloků, které slouží k ovládání zařízení a čtení stavových informací.

U flash disků probíhá komunikace na základě tzv. bulk-only protokolu [1]. Výměna informací je tímto protokolem definována ve třech etapách. První etapou je přenos příkazu, následuje výměna dat a poslední je přenos stavu.

Použitý protokol je potřeba vyznačit specifikováním hodnoty bInterfaceProtocol v deskriptoru. Protokolu bulk-only náleží hodnota 50h.

Přenos příkazu

Příkazy jsou v bulk-only protokolu přenášeny v přesně definované struktuře zvané CBW (Command-Block Wrapper). Tato struktura uchovává položky uvedené v tabulce 2.3.

Hodnota CBWCB, která představuje příkaz, závisí na zvolené příkazové sadě. Příkazovou sadu určuje podtřída, uvedená v deskriptoru jako bInterfaceSubClass. Deskriptory jsou popsány ve zdrojovém souboru `usb_slave_boms.c`, dostupném na přiloženém CD v adresáři `/firmware/keylogger/src`. Podle [3] se pro standardní flash disky používá podtřída 06h, která určuje příkazovou sadu SCSI.

Výměna dat

Data jsou odesílána pomocí datových bulk endpointů. Směr komunikace určuje použitý endpoint (IN/OUT), jedná se o přenosy typu bulk.

Název hodnoty	Počet bitů	Popis
dCBWSignature	32	Konstantní hodnota 43425355h, která identifikuje CBW
dCBWTag	32	Hodnota, která přiřazuje příkaz příslušnému CSW
dCBWDataTransferLength	32	Počet bytů, který bude přenesen při výměně dat
bmCBWFlags	8	Bit 7 určuje směr přenosu (1=k hostiteli, 0=od hostitele), zbytek nulové bity
Rezervováno	4	Nuly
bCBWLUN	4	Číslo LUN pro zařízení s více logickými jednotkami
Rezervováno	3	Nuly
bCBWCBLength	5	Délka následujícího příkazu v bytech (1-16)
CBWCB	128	Předávaný příkaz pro zařízení

Tabulka 2.3: Popis struktury CBW, přejato z [1]

Přenos stavových informací

Datový přenos je zakončen odesláním struktury CSW (Command Status Wrapper). Položky struktury tak, jak následují za sebou uvádí tabulka 2.4.

Název hodnoty	Počet bitů	Popis
dCSWSignature	32	Konstantní hodnota 53425355h, která identifikuje CSW
dCBWTag	32	Hodnota, která byla uvedena v příslušejícím CBW
dCSWDataResidue	32	Rozdíl mezi počtem očekávaných bytů a počtem skutečně odeslaných bytů
bmCSWStatus	8	00h = příkaz zpracován úspěšně 01h = chyba při zpracování příkazu 02h = fázová chyba

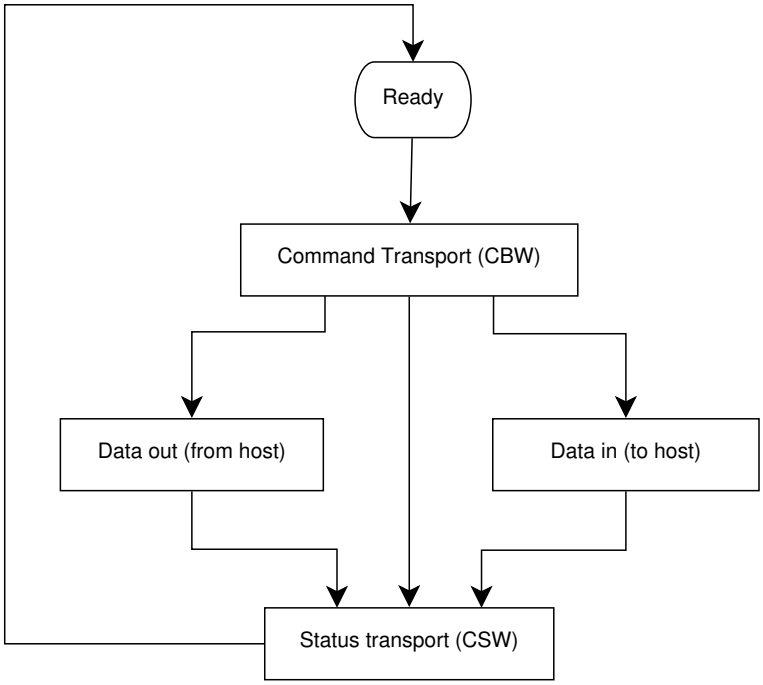
Tabulka 2.4: Popis struktury CSW, přejato z [1]

Schéma komunikace uvádí obrázek 2.4. Více podrobností o bulk-only protokolu je možné dohledat také v [17].

2.7 SCSI příkazy v zařízeních BOMS

Následující část čerpá z [2], [16] a [13].

SCSI (Small Computer Systems Interface) je standardizované rozhraní, které definuje sadu příkazů. Tyto příkazy slouží pro komunikaci mezi periferním a hostitelským zařízením.



Obrázek 2.4: Diagram komunikace při použití protokolu bulk-only, přejato z [17]

2.7.1 INQUIRY

Základním příkazem, který musí zařízení tohoto typu podporovat je SCSI příkaz INQUIRY. Výstupem tohoto příkazu je struktura popisující zařízení.

Množina příkazů, které musí zařízení podporovat, se odvíjí od typu zařízení a verze příkazové sady, které jsou specifikovány právě v odpovědi na příkaz INQUIRY. Podle [2] používají nejčastěji standardní flash disky v poli typ zařízení hodnotu 00h (blokové zařízení s přímým přístupem) a jako verzi příkazové sady hodnotu 04h pro SPC-2 (SCSI Primary Commands 2).

V následujících odstavcích jsou rozebrány příkazy, které je nutné implementovat pro správnou funkčnost zařízení.

2.7.2 MODE SENSE

Tento příkaz je označován jako volitelný, avšak operační systémy Windows vyžadují jeho podporu. Po vyvolání je očekávána struktura obsahující informace uvedené v tabulce 2.5.

2.7.3 REQUEST SENSE

Příkaz slouží ke čtení stavových informací. Je využíván zejména při detekci chyb, jako rozšiřující informace o typu způsobené chyby. Struktura dat je uvedena v tabulce 2.6.

Název hodnoty	Počet bitů	Popis
MODE DATA LENGTH	8	Délka následujících dat v bytech (zde 3)
MEDIUM TYPE	8	Typ média, zde 00h pro bloková zařízení
DEVICE-SPEC. PARAMETER	8	Bit 7: 1 pokud médium je zamknuté proti zápisu bit 6..5: rezervováno bit 4: 1 pokud zařízení podporuje cachování bit 3..0: rezervováno
BLOCK DESC. LENGTH	8	Délka následujících parametrů (zde 0)

Tabulka 2.5: Popis struktury odpovědi na příkaz MODE SENSE, přejato z [2]

2.7.4 TEST UNIT READY

Hostitel použije tento příkaz ke zjištění, jestli je zařízení již připraveno vykonávat následující příkazy. Pokud není připraveno, je v poli bmCSWStatus bloku CSW (2.4) nastavena hodnota 01h a jsou nastaveny parametry sense data (2.6), indikující případnou chybu.

Tento příkaz nemá fázi výměny dat.

2.7.5 READ

Příkaz READ slouží ke čtení dat ze zpřístupňovaného média. Podle velikostí operandů příkazu je možné rozlišit více variant. Variantu příkazu udává číslo v závorce za jménem příkazu. Pro READ(10) je typická délka 10B, ze které jsou 4B vyhrazeny pro adresu LBA a 2B pro délku přenosu. Struktura příkazu READ(10) je uvedena v tabulce 2.7.

2.7.6 READ CAPACITY

Příkaz READ CAPACITY slouží k informování hostitele o celkové kapacitě zpřístupňovaného média. Příkaz ve fázi datové výměny protokolu bulk-only přenáší strukturu dvou hodnot, jejíž formát je popsán v tabulce 2.8.

2.7.7 READ FORMAT CAPACITIES

Příkaz READ FORMAT CAPACITIES slouží k vyčtení dat nutných pro formátování média. Struktura reakce je uvedena v tabulce 2.9. Uvedená struktura v tabulce je minimální, tzn. obsahuje pouze jediný (povinný) deskriptor momentální/maximální kapacity.

2.7.8 WRITE

Příkaz WRITE slouží k zápisu bloků na médium. Varianta příkazu WRITE(10) má shodnou strukturu parametrů s příkazem READ(10), viz tabulka 2.7. Rozdíl je pouze v operačním kódu, který má pro tento příkaz hodnotu 2Ah. Položky LBA určují počátek zapisovaných dat a TR. LENGTH určuje počet zapsaných bloků.

Název hodnoty	Počet bitů	Popis
VALID	1	1 pokud pole INFORMATION obsahuje platné informace
RESPONSE CODE	7	70h pro aktuální chyby, 71h pro pozdržené chyby (využíváno při použití cache)
Zastaralé	8	
FILEMARK	1	Používáno pro proudová zařízení
EOM	1	Konec média
ILI	1	Indikátor nesprávné délky (použ. s READ LONG, WRITE LONG a proudovými READ příkazy
Rezervováno	1	
SENSE KEY	4	Informace popisující chybu
INFORMATION	32	Informace specifické pro příkaz nebo zařízení
ADDITIONAL SENSE LEN	8	Počet rozšiřujících bytů, které následují za touto položkou
COMMAND-SPEC. INFORM.	32	
ASC	8	Poskytuje rozšiřující informace o chybě (jinak 00h)
ASCQ	8	Další informace k ASC, nast. na 00h pokud nevyužito
FIELD REPLAC. UNIT CODE	8	Identifikuje chybnou komponentu, nast. na 00h, pokud není známa
SENSE KEY SPECIF.	24	Pokud 7. bit prvního bytu je 1, zbytek pole obsahuje platnou informaci
Rozšiřující data	0..n	Rozšiřující stavové byty (spec. pro určité výrobce)

Tabulka 2.6: Popis struktury odpovědi na příkaz REQUEST SENSE, přejato z [2]

Název hodnoty	Počet bitů	Popis
OP. CODE	8	Pro READ(10) 28h
Nepoužité	8	Nepoužité, nast. na 0
LBA	32	Adresa logického bloku, počínaje MSB
Nepoužité	8	Nepoužité, nast. na 0
TR. LEN	16	Počet logických bloků k přenosu, počínaje MSB
CONTROL	8	Rozšiřující parametry

Tabulka 2.7: Popis struktury příkazu READ(10), přejato z [13]

Název hodnoty	Počet bitů	Popis
Adresa LBA nejvyšší adresy	32	Adresa nejvyššího bloku je rovna počtu bloků - 1 (adresace od nuly)
Délka bloku	32	Délka jednoho bloku v bytech (standardně 512)

Tabulka 2.8: Popis struktury příkazu READ CAPACITY, přejato z [2]

Název hodnoty	Počet bitů	Popis
Rezervováno	24	
CAP. LIST LEN.	32	Délka násled. deskriptorů n * 8 kde 0 < n < 32
NUM. OF BLOCKS	32	Počet bloků (nikoliv adresa posledního), MSB prvně
Rezervováno	6	
DESC. TYPE	2	00b = rezervováno 01b = hodnota je max pro nenaform. medium 10b = hodnota je aktuál. vel. naform. média 11b = médium není přítomno
BLOCK LEN.	32	Počet bytů v bloku

Tabulka 2.9: Popis struktury příkazu READ FORMAT CAPACITIES, přejato z [16]

Kapitola 3

SD karty

Následující kapitola čerpá z [12], [15], [26] a [25].

SD (Secure Digital) je formát paměťové karty. Jedná se o nevolatilní paměť, tzn. paměť, která je schopna uchovávat informace i bez napájení. Celkem můžeme rozlišit tři rodiny karet, jedná se o SDSC (Standard-capacity), označované také pouze jako SD, SDHC (High-capacity) a SDXC (Extended-capacity).

V každé rodině existují ještě tři možné typy pouzder karet. Jedná se o standardní velikost, velikost *mini* a velikost *micro*. Přehled typů SD karet je uveden v tabulce 3.1.

	SDSC	SDHC	SDXC
Kapacita	do 2GB	od 2GB do 32 GB	od 32GB do 2TB
Rychlost	NS (Normal-speed), HS (High-speed)	NS, HS, UHS-I,	NS, HS, UHS-I

Tabulka 3.1: Přehled typů SD karet

Přístup k plné specifikaci SD je zpoplatněn, tato specifikace mj. dopodrobna popisuje vícekanálovou komunikci s využitím SD Bus. V roce 2006 SD Asociace zveřejnila zjednodušenou verzi specifikace v [12].

3.1 Struktura SD karty

Každá karta obsahuje sadu povinných řídících registrů. Jedná se o registry OCR, CID, CSD, RCA, DSR a SCR. Tyto registry jsou přístupné pomocí příkazů karty, příkazy interpretuje tzv. *Card Interface Controller*, který je rovněž připojen k paměťovému jádru karty. Přehled registrů je uveden v tabulce 3.2.

3.2 Stavy karty

Karta se během provozu nachází v různých stavech. Prvním stavem je vypnuto. Jde o případ, kdy je karta odpojena, příp. napájení nepřesahuje 0,5V.

Dále můžeme rozlišit identifikační režim, během kterého je karta vyresetována, zkontrolován rozsah napětí a karta je dotázána na svou RCA (Relative Card Address). Během

Název registru	Šířka (bitů)	Stručný popis
OCR	32	Operation Conditions Register
CID	128	Card Identification Number
CSD	128	Card Specific Data (informace o přístupu k obsahu karty)
RCA	16	Relative Card Address (adresa karty, která je vyzvednuta během inicializace)
DSR	16	Driver Stage Register (nastavení výstupních parametrů)
SCR	64	SD Configuration Register (informace o vlastnostech karty)

Tabulka 3.2: Základní přehled registrů SD karty, přejato z [12]

tohoto procesu je karta taktována frekvencí, která je definována pro inicializaci. V tomto režimu může být karta ve stavu idle, ready a identification.

Vzhledem k tomu, že postupem času docházelo k rozšíření vlastností SD karet, je proces inicializace komplikován testy, o kterou verzi se jedná.

Posledním operačním režimem je režim datového přenosu. Po jeho zahájení je možné s kartou komunikovat na vyšší frekvenci. Host (master) odešle příkaz pro zjištění obsahu CSD registru (SEND CSD), tím zjistí např. délku bloku, kapacitu apod. Volitelným broadcastovým příkazem SET DST jsou následně nastaveny vhodné parametry přenosu připojených karet. Následně je možné zvolenou jednu kartu pomocí přkazu CMD7 převést do stavu transfer. S touto kartou následně probíhají datové operace. Ostatní neadresované karty se přepnou do stavu stand-by. Celkem tedy v operačním režimu můžeme rozlišit tyto stavy: stand-by, transfer, sending, receive, programming a disconnect.

Při úspěšném dokončení zápisového cyklu karta přechází do stavu programming, pokud byl přenos úspěšný, popř. do stavu transfer, pokud při přenosu došlo k chybě. Během stavu programming není možné provádět zápisové ani čtecí operace. Zápisové i čtecí přenosy mohou být kdykoliv ukončeny příkazem STOP COMMAND (CMD12).

3.3 Komunikace s kartou

Komunikace s kartou probíhá pomocí příkazů. Příkaz je dlouhý 48 bitů a přenos začíná nejlevějším (MSB) bitem. Na příkazy karta odpovídá pomocí odezev.

Po připojení je karta napájena standardně napětím 3,3V. Novější typy karet (SDHC a SDXC) mohou být povozovány i při napětí 1,8V, změně napětí ale musí předcházet příkaz, podle kterého hostitel zjistí dostupnost tohoto režimu a uvědomí kartu. Inicializace může probíhat v 1-bitovém synchronním režimu nebo v režimu SPI. Hostitel má možnost režim vybrat pomocí napětí na 1. pinu karty během zapínání. Tento režím nelze nadále měnit bez restartu celého komunikačního procesu.

Po inicializaci lze zvýšit kmitočet synchronní komunikace a v případě komunikace přes SD Bus i počet komunikačních kanálů až na čtyři. Hostitel pomocí příkazu zjistí maximální počet používaných paralelních komunikačních kanálů. Přenos dat po více komunikačních kanálech je rozdělen po jednotlivých bitech, při použití čtyř kanálů se tak v jednom taktu přenesou čtyři bity, které v datovém bloku následují za sebou. Vodič SPSCK přenáší hodinový signál směrem od mastera k zařízením typu slave.

3.4 Rozhraní SPI

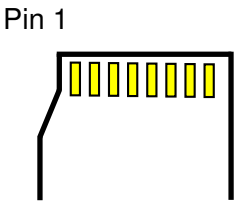
Odstavec čerpá ze zdrojů [4] a [27].

Rozhraní SPI (Serial Peripheral Interface) bylo vyvinuto pro komunikaci mezi hlavním zařízením, nejčastěji mikrokontrolérem a periferními zařízeními. Lze využít pro spojení point-to-point, lze zapojit i jako jednoduchou sběrnici.

Zařízení připojené do sběrnice SPI [6] se nachází v jednom z režimů master nebo slave. Zařízení typu master smí být ve sběrnici pouze jedno, generuje hodinový signál, od kterého se odvíjí komunikace. Zařízení typu slave představují nejčastěji periferie, které komunikují s masterem. Master určuje adresované zařízení pomocí vodiče SS (Slave Select). Komunikace je založena na principu posuvných registrů, které jsou propojeny vodiči MISO (Master In Slave Out) a MOSI (Master Out Slave In). Za osm taktů tak proběhně výměna jednoho bytu obsahů posuvných registrů, jedná se tedy o komunikaci full-duplex, kdy v každém taktu je přenesen právě jeden bit oběma směry. Posuvné registry jsou taktovány právě signálem SPSCK, po osmi taktech tedy dojde k výměně jednoho bytu. Při jednosměrné komunikaci jeden z komunikujících odesílá "bezvýznamné" bity.

3.5 Použití SPI pro komunikaci s kartou

Komunikace s použitím rozhraní SPI je bytově orientována, velikost datových bloků je násobkem osmi bitů. Komunikační proces je zahajován hostitelem tím, že nastaví nízkou úroveň na pin 2. V případě užití SPI pro komunikaci je potřeba řešit výběr adresované karty. Pro komunikaci je použit pin 3, jako vodič přenášející data směrem do karty, pin 5, který určuje taktování komunikace SPI a pin 7, přenášející informace směrem z karty do hostitele. Kromě zmíněných je v tomto režimu dále využit pin 6, který je uzemněn a pin 4, který je přiveden na napájecí napětí. Režim SPI je vhodný pro méně náročné aplikace, kde není vyžadována velká šířka datového přenosu, jeho výhodou jsou především nižší nároky na hostitele. Ve většině případů jde o mikrokontrolér, ten bývá běžně vybaven právě rozhraním pro komuniaci pomocí SPI. Nevýhodou je, že v režimu SPI nejsou dostupné některé příkazy. Napájecí napětí v tomto režimu je fixně 3,3V, [12].



Obrázek 3.1: Obrázek vyznačuje pořadí pinů na microSD kartě, přejato z [31]

Možné zapojení a významy vodičů SD karty při zapojení pomocí SPI jsou shrnuty v tabulce 3.3, pořadí pinů vysvětluje obrázek 3.1.

Pin	Význam v režimu SPI	Popis
1	Nepoužito	
2	CS	Výběr adresované karty
3	MOSI	Datový vstup do karty
4	Napájení - Vdd	3,3V
5	CLK	Hodinový signál pro synchronizaci komunikace
6	Napájení - Vss	
7	MISO	Datový výstup z karty
8	Nepoužito	

Tabulka 3.3: Přehled významu pinů SD karty v režimu SPI, převzato z [31]

Kapitola 4

Vinculum VNC2

Následující sekce čerpá ze zdroje [7] a [9].

Vinculum VNC2 je 16-bitový MCU se dvěma nezávislými USB porty host/slave. Obsahuje 256KB flash pamět pro ukládání programu a 16KB RAM. Ze strany výrobce je poskytováno široké spektrum knihoven a příkladů jak s nimi pracovat, což značně usnadňuje vývoj.

4.1 Operační systém Vinculum

Firmware navrhovaný pro mikrokontroléry Vinculum VNC2 může být založen na tzv. VOS (Vinculum Operating System).

Model takového firmware sestává ze tří vrstev. Na nejnižšní úrovni je vrstva VOS kernel, další vrstvou jsou FTDI ovladače a FTDI knihovny. Nejvyšší vrstva implementuje uživatelskou aplikaci. Ta pomocí API funkcí komunikuje s ovladači a přistupuje tak k HW zdrojům.

4.1.1 VOS kernel

VOS kernel spravuje HW zdroje, přerušení a plánování. Je to preemptivní multitaskingové jádro OS.

VOS obstarává prioritně založené plánování. Úkoly jsou ohodnoceny prioritami, na jejichž základě se plánovač rozhoduuje, která úloha dostane přidělen výpočetní výkon. Doba, kterou dostane úloha přidělenu k běhu, je založena na hodnotě tzv. kvanta. Po uplynutí tohoto času je úloha pozastavena a její kontext je uložen, následovně je spuštěna další úloha ze seznamu připravených úloh.

Synchronizace úloh

K synchronizaci úloh jsou k dispozici prostředky jádra. Jedná se o mutexy k zajištění výlučného přístupu ke zdroji a semafory pro synchronizaci vláken. Kritické sekce slouží k vymezení úseků kódu, ve kterých nesmí dojít k přepnutí plánovačem.

Device manager

Přístup k vrstvě ovladačů HW z kódu uživatelské aplikace je možný přes tzv. device manager. Po otevření přístupu k zařízení device manager alokuje tzv. device handler, pomocí

kterého je možné se odkazovat k otevřenému zařízení. Otevření zařízení zajistí výlučný přístup. Device manager je přístupný pomocí následujících funkcí:

`vos_dev_open()` Otevření zařízení, vytvoření device handleru

`vos_dev_close()` Zavření zařízení

`vos_dev_read()` Čtení ze zařízení

`vos_dev_write()` Zápis do zařízení

`vos_dev_ioctl()` Poslání příkazu zařízení

Inicializace jádra

Interní datové struktury jádra a parametry plánovače jsou inicializovány funkcí `vos_init()` před spuštěním jádra.

Důležitým parametrem plánovače úloh je tzv. kvantum. Jde o počet jaderných cyklů, po jejichž dobu bude běžet daná úloha. Výchozí hodnotou je kvantum 50, tato hodnota je vhodná jak pro občasné výpočty, tak pro interakci s okolím. Snížením této hodnoty lze docílit častějšího přepínání úloh, ovšem za cenu vyšší režie jádra. Běžící úloha může být pozastavena dříve, než udává hodnota kvanta. Může jít o případ přerušení SW či HW, blokování, kdy dané vlákno zavolá funkci jádra, která je blokující do úplného dokončení (např. `vos_lock_mutex()`), přeplánování, kdy mohlo právě dojít k odblokování úlohy s vyšší prioritou pomocí mechanismu přerušení nebo došlo k zavolání funkce typu "delay".

Vlákna

Vytvoření vlákna zajišťuje `vos_create_thread()`. Každému vláknu je automaticky touto funkcí dynamicky přiřazena oblast paměti pro zásobník a paměť pro uložení stavu. Počet souběžných vláken je limitován zajména velikostí paměti. Každému vláknu je přiřazena priorita z intervalu 0 až 31, ačkoliv nula je rezervována pro idle task. Vlákna se stejnou prioritou jsou spouštěna po sobě metodou round-robin. Obecně je doporučeno nepoužívat metodu polling bez užití synchronizačních mechanismů popř. "delay" funkcí. V opačném případě totiž dojde k znemožnění běhu vláken s nižší prioritou, což ve výsledku způsobí deadlock systému.

K běhu vláken dochází se spuštěním plánovače funkcí `vos_start_scheduler()`.

4.1.2 FTDI ovladače

Pro řízení perfierií dostupných na čipu lze využít HW ovladače, které poskytuje sám výrobce. Tyto ovladače pracují nad VOS. K těmto ovladačům výrobce poskytuje navíc i tzv. funkční ovladače, které implementují pomocí HW ovladačů konkrétnější funkčnost, např. čtení dat z klávesnice. Náleduje popis ovladačů důležitých z pohledu implementace USB keyloggeru.

SPI ovladač

Taktování SPI v režimu master i slave je podporováno až do jedné čtvrtiny frekvence taktování CPU. Nejnižší frekvence pro SPI master je 1/256 frekvence taktování CPU. V režimu master je nutné před každým čtením provést zápis stejné délky, jako jsou očekávaná data.

Tento fakt vychází ze specifikace rozhraní SPI, [6]. K hodinovým pulzům synchronizačního signálu rozhraní dochází pouze v případě zápisu (odesílání dat), zápis tak způsobí načtení aktuálních dat do bufferu. Operace čtení je blokující dokud neproběhne načtení požadované délky dat do bufferu. Po provedení několika zápisových operací jsou přijatá data postupně ukládána do přijímacího bufferu. Takto uložená data lze následně zpracovat operacemi čtení.

USB slave

Jedinou instancí ovladače je možné ovládat USB port 1, USB port 2 nebo oba vestavěné USB porty. Ovladač si udržuje kontext pro případné oba porty. Port má přidělen svůj handle, přes který je možné se na něj odkazovat. Pro jednoznačnou identifikaci endpointu je vyžadována kombinace jedinečného `VOS_DEVICE` handle a `usbslave_ep_handle_t`. Přístup k USB portu je v režimu slave založen na přístupu k endpointům portu. Ovladač slave USB nemůže být za chodu přenastavován.

Pro správnou enumeraci je nutné zajistit následující postup:

1. Čekání na přijetí SETUP paketu
2. Dekódování přijatého paketu
3. Obsloužení následujících dotazů:
   - Změna adresy USB slave
   - Konfigurace
   - Získání deskriptorů
4. Obsluha dotazu pro danou třídu zařízení (device class)
5. Obsluha dotazu pro zařízení daného výrobce (vendor id)
6. Načtení případných dalších dat SETUP paketu
7. Potvrzení příjmu ACK paketem

USB host

Ovladačem USB host je možné spravovat libovolný jeden port, nebo oba přítomné USB porty, stejně jako u USB slave ovladače. Navíc však USB host ovladač udržuje v paměti datovou strukturu reprezentující USB slave rozhraní připojená k tomuto portu. Tuto strukturu lze prohledávat na zařízení určitého typu. Ovladač není možné přenastavovat za běhu.

Celkem jsou na VNC2 přístupné dva root huby, z nichž každý je spjat s jedním USB portem, pokud se tento port nachází v režimu USB host. Po připojení zařízení k root hubu je zahájen proces enumerace, při odpojení jsou uvolněny reference spjaté s tímto zařízením. Ke kořenovému rozbočovači je možné připojit hub s dalšími zařízeními.

Komunikace se odehrává na základě datové struktury popisující transakci. Tato struktura se nazývá transfer block a je předávána jako parametr funkcím `vos_dev_read()` a `vos_dev_write()`. Obsahuje cílový endpoint, samotná data, délku dat, návratový kód a semafor, který se využívá pro hlášení stavu uživatelské aplikaci, případně pro blokaci transakce.

Kapitola 5

Návrh a realizace firmware

Následující kapitola se zaměřuje na návrh a implementaci firmware pro USB keylogger. Zvolený MCU VNC2 je poskytován s poměrně širokou škálou knihoven a ovladačů, které je možné v následující implementaci využít. Tyto knihovny jsou odladěny přímo pro daný HW a jejich použití je snadné.

Pro průběžné testování a ladění SW byla použita vývojová deska V2-EVAL (Rev1) [8] s modulem V2-EVAL-EXT64.

Daný úkol byl dekomponován na menší problémy, kterým jsou věnovány následující odstavce.

5.1 HID klávesnice

5.1.1 Slave klávesnice

Prvním dílčím úkolem bylo použít vývojovou desku k vytvoření zařízení, které by bylo schopno se po zasunutí do počítače chovat jako klávesnice, jednotlivé stisky kláves by bylo ovšem možné generovat z vnitřního programu MCU.

Ovladač slave klávesnice

Firma FTDI nabízí pro MCU VNC2 také ovladač, který má zmíněnou funkčnost. Ovladač se nachází v souborech `USBSlaveHIDKbdDrv.c` a `USBSlaveHIDKbdDrv.h`, které jsou součástí příkladů po instalaci vývojového prostředí Vinculum II IDE [10]. V těchto souborech se nachází rovněž definice deskriptorů klávesnice (více o USB deskriptorech je napsáno v části 2.4). Následující text vychází ze zdrojového kódu USB slave ovladače klávesnice, dostupného po instalaci nástroje Vinculum II IDE.

Ovladač USB slave klávesnice zajišťuje veškerou režii komunikace, stará se o správnou enumeraci zařízení apod. Stačí jej pouze inicializovat funkcí `usbslavehidkbd_init(VOS_DEV_NUM)`, potom ovladač standardním způsobem otevřít pomocí device manageru (který je popsán v 4.1.1) a voláním IOCTL `VOS_IOCTL_USBSLAVEHID_ATTACH`, při kterém je jako parametr předáván handle na USB slave port, začít enumeraci.

Po enumeraci je možné standardním způsobem device manageru zapisovat data, která jsou odesílána jako stisky kláves do počítače. Čtení dat z počítače v případě klávesnice dává smysl snad jen u LED diod signalizujících zapnutí Scroll Lock, Caps Lock a Num Lock. V tomto ovladači ale není podporováno. Proto je nutná drobná úprava zdrojového kódu, která je blíže popsána v 5.1.3.

Formát přenosu dat

Propagace stisknutých kláves z klávesnice do počítače se provádí zápisem osmi bytů, kde druhý až osmý byte představuje tzv. *scancode* právě držených kláves. Scan kódy použité v USB HID jsou odlišné od těch používaných v klasických klávesnicích s PS/2 (mini-DIN) konektorem. Přehled scan kódů je k dispozici např. v [29]. První byte report paketu slouží k přenosu příznaků, jako je např. stisknutý Caps Lock.

Maximum současně držených kláves, které lze detekovat, je dáno počtem bytů v report paketu určených k uchování scan kódů, zde 7. Formát report paketů je udáván report deskriptorem.

5.1.2 Host klávesnice

Dalším úkolem, který se nabízí, je zajistit, aby zařízení, které představuje vývojová deska V2-EVAL, mohlo číst data z klávesnice připojené do USB portu, který je v režimu host.

Pro tento účel nabízí firma FTDI rovněž již předpřipravený ovladač. Tentokrát ve formě kompilované knihovny.

Ovladač host klávesnice

Pro použití knihovny s ovladačem je nejprve nutné ji připojit ve vývojovém prostředí Vinculum II IDE tak, aby byla ve fázi linkování připojena do výsledného souboru s firmwarem zařízení. To je možné provést v záložce `Build` tlačítkem `Libraries`.

Ovladač klávesnice v režimu host je nutné inicializovat před spuštěním plánovače VOS funkcí `usbHostHID_init(VOS_DEV_NUM)`. Ovladač je potom potřeba otevřít pomocí funkcí device manageru a připojit.

Pokud byl při zakládání projektu použit App Wizard prostředí Vinculum II IDE a tento ovladač byl vybrán, přidají se do kódu i další pomocné funkce `hid_attach()` a `hid_detach()`, které zapouzdřují běžné akce otevření, připojení a odpojení ovladače.

Podle vzorového příkladu práce s ovladačem USB host klávesnice, který je rovněž součástí Vinculum II IDE, je po připojení ovladače a před zahájením operace čtení potřeba vykonat následující akce. První je načtení report deskriptoru z připojené klávesnice. Poté odeslání příkazu set idle a get idle. Ke všem těmto akcím jsou připravené rutiny právě v knihovně s ovladačem, takže jde o pouhé volání funkce IOCTL device manageru s patřičným parametrem.

Čtení dat z připojené klávesnice

Formát čtených dat je shodný s formátem zápisu dat popsaným v odstavci 5.1.1. Délka čtených dat je rovněž osm bytů.

5.1.3 Přemostění USB slave a host portů klávesnice

Pro správnou funkčnost je nutné docílit propojení obou ovladačů tak, aby data z klávesnice transparentně procházela zařízením beze změny. Tak bude možné po zapojení zařízení mezi počítač a USB klávesnici nechat data nepozorovaně proudit přes zařízení a přitom nezávisle zpracovávat také na straně počítače.

Dalším možným řešením by mohlo být propojení datových vodičů obou USB portů přímo. Poté pomocí MCU, který by byl zapojený paralelně s počítačem, analyzovat protékající data. Takové zařízení by bylo možné označit za hardwarový analyzátor USB paketů.

Tento způsob realizace by ovšem komplikoval splnění požadavku na možné zpřístupnění úložiště dat, kam probíhá záznam stisků kláves. Navíc by šlo o nestandardní využití zvoleného MCU. Proto byl zamítnut.

Realizace propojení ovladačů

USB host i slave ovladače klávesnice předávají/vyžadují data ve formě nedotčených osmibytových report paketů. Proto je možné přemostění obou USB portů realizovat obyčejným cyklem, který je spuštěn ve vyhrazeném vlákně `thread_bridge()`. Jde vpodstatě o neustálé čtení dat z klávesnice a předávání dat počítači. To ale nezajistí také správné rozsvěcení LED diod signalizujících např. stav příznaku Caps Lock. Proto je nutná drobná úprava USB slave ovladače klávesnice.

Realizace propojení stavu LED diod

Úprava slave ovladače spočívá v přidání možnosti reakce na přijatá data, která souvisí se stavem LED diod na klávesnici. Jde o přidání 32 bytového ukazatele na funkci přímo do kontextu ovladače. Pomocí IOCTL volání `VOS_IOCTL_USBSLAVEHID_SET_REPORT_FUNC` je při inicializaci nastaven tento ukazatel tak, aby pomocí něj bylo možné správně volat funkci pro nastavení stavu LED diod na klávesnici připojené k USB host ovladači. Tato funkce se nazývá `usbslave_process_report()` a nachází se v souboru `threads_keylogging.c`.

Funkce `usbslave_process_report()` přebírá jeden jednobytový parametr, který obsahuje informace o nastavených příznacích LED diod. Formát těchto dat je shodný s formátem report paketů očekávaných dat na straně klávesnice.

USB slave ovladač klávesnice po přijetí požadavku se stavem LED diod zavolá zmíněnou funkci a předá ji jako parametr jednobytovou hodnotu přijatou v tomto požadavku. Přeposílání stavu LED diod je potom také zcela transparentní.

5.2 Zpracování protékajících dat

Aby bylo možné zaznamenávat stisknuté klávesy, bylo by nejdříve vhodné detekovat jednotlivé stisky jako události a navíc je převést do srozumitelnější podoby, než jsou, po sběrnici přenášené, scan kódy.

Zpracování se odehrává v dalším vyhrazeném vlákně (společně se zápisem, viz 5.3), pojmenovaném `thread_keydec()`.

5.2.1 Detekce stisknutých kláves

USB HID klávesnice má celkem 231 scan kódů [29], takže pro reprezentaci jejich stavu příslušejících kláves stačí v paměti 32 bytová proměnná.

Pro detekci událostí kláves jsou v paměti drženy vždy aktuální a předchozí snímek stavu kláves. Jedná se o dva 256 bitové vektory (32B). Procházením přijatého report paketu z klávesnice připojené do USB host portu dojde k nastavení jednotlivých bitů na příslušná místa aktuálního vektoru. Následně je aktuální a předchozí vektor prohledán na změny. Prohledávání vektorů se odehrává po bytech. Teprve po najití neshody je patřičný byte prohledáván na změny v jednotlivých bitech. Změna bitu poté v konstrukci `switch(keystatus)` vyvolá patřičnou funkci `keydown()` příp. `keyup()`.

Událostí se rozumí stisk, příp. puštění klávesy, které jsou reprezentovány konstantami `KEYDOWN` a `KEYUP` v konstrukci `switch`.

Podobně jsou zpracovávány příznaky přenášené v prvním bytu USB report paketu.

Pomocné funkce pro práci s bitovým snímkem klávesnice, detekci událostí a převodními tabulkami jsou implementovány v souborech `scancode.c` a `scancode.h`.

Implementace `keydown()` a `keyup()` se nachází v souboru `keypress.c` a `keypress.h`. Funkce přebírají celkem tři parametry. Scan kód potřebný ke svázání probíhající události s příslušnou klávesou. Dále potom příznakový byte (prní byte posílaný v USB report paketech), důležitý pro možnost zpracování také událostí týkajících se kláves CTRL apod. Poslední parametr právě určuje, zda se jedná o událost spjatou s běžnou klávesou nebo příznakovou klávesou CTRL apod.

Jak již bylo zmíněno, události spjaté s běžnými klávesami jsou zpracovávány odděleně (ale však stejným způsobem), je nutné uchovávat dvě převodní tabulky. Převodní tabulkou se zde rozumí pole, které obsahuje ukazatele na ASCII řetězce. Ty jsou uspořádány vzestupně, podle čísel scan kódů tak, že scan kód lze použít pro indexaci hledaného odpovídajícího řetězce v tabulce. Většinou tabulka obsahuje jednoznakové řetězce, avšak nutnost ukládat řetězce namísto pouhých znaků je způsobena názvy některých kláves (enter, backspace apod.).

Převodní tabulka byla inspirována rozložením klávesnice US QWERTY.

5.2.2 Problémy a omezení při detekci kláves

Dlouhodobé držení klávesy

Při stisku klávesy dojde na počítači k detekci události a zpracování. Pokud je ovšem klávesa držena nepřetržitě, dojde po určité chvíli ke znovudetekcím stisku, tektokrát rychle následujícím po sobě. Toto chování je implementováno na straně počítače a není možné v USB keyloggeru zpětně detekovat, za kolik běžných stisků bylo jedno dlouhé držení klávesy interpretováno. Z výše uvedeného plyne omezení pro představené a implementované řešení, které tak i dlouhé držení klávesy detekuje jako jeden stisk.

Zajímavým rozšířením by mohlo být doplnění informace, jak dlouho byla daná klávesa držena např. pomocí časových značek, které by se zaznamenávaly spolu s událostí.

Rozložení kláves na straně počítače

Lokalizované rozložení klávesnice se rovněž odehrává na straně operačního systému počítače, proto není možné zpětně detekovat, jaké rozložení je nastaveno, a jaké rozložení tedy použít pro převod scan kódů do čitelné formy, která je zařízením keylogger zaznamenávána. Převod scan kódů do čitelné podoby se tedy odehrává na základě pevně dané tabulky rozložení US QUERTY.

Zajímavým rozšířením by mohlo být implementovat možnost načtení převodové tabulky z ukládací paměti (zde microSD karty). Ve výsledném řešení je ale ponechána možnost převodní tabulky měnit pouze ručně, ve zdrojovém souboru `scancode.c`.

5.3 Ukládání záznamů

Pro možnost přístupu na SD kartu naformátovanou souborovým systémem FAT nabízí firma FTDI již hotové řešení ve formě ovladačů SPI master, SD, a FAT. Řešení je tedy postaveno na schopnosti SD karet komunikovat po sběrnici SPI, popsané v 3.5.

5.3.1 Použití ovladačů SPI master a SD

Inicializace ovladače SPI master je provedena před spuštěním plánovače systému VOS. Pro správnou funkčnost je nutné nastavit velikost bufferu na 512 bytů.

SD ovladač je inicializován funkcí `sd_init(VOS_DEV_NUM)` rovněž před spuštěním plánovače VOS. Po spuštění plánovače je nadále nutné pomocí IOCTL zprávy `MSI_IOCTL_SD_CARD_ATTACH` propojit ovladače SD karty a SPI.
Zprávou `MSI_IOCTL_SD_CARD_INIT` je karta přepnuta do SPI režimu a připravena pro přenos dat.

5.3.2 Použití ovladače FAT

Přenos dat se odehrává za režie ovladače souborového systému FAT. Ten je rovněž dodáván s vývojovým prostředím Vinculum II IDE a je plně kompatibilní s ovladačem SD.

Před spuštěním plánovače VOS je ovladač FAT inicializován funkcí `fatdrv_init(VOS_DEV_NUM)`. IOCTL zpráva `FAT_IOCTL_FS_ATTACH` připojí k ovladači FAT handle ovladače SD a předá číslo partition, na kterou se daná instance ovladače váže.

Nakonec je nutné tuto instanci ovladače systému souborů prohlásit za výchozí, používanou při operacích se soubory. To zajistí volání `fsAttach(fathandle)`. Poté je možné začít používat funkce `fopen()`, `fwrite()` atd.

5.3.3 Optimalizace ukládání záznamů

Ve zmíněných funkcích `keydown()` a `keyup()` by bylo možné použít záznam stisků kláves přímo, za použití zmíněných funkcí pro práci se soubory. To by ale značně prodlužovalo reakční dobu na stav klávesnice kvůli delší době zápisu. Proto bylo vhodné systém do jisté míry optimalizovat.

Návrh optimalizace

Optimalizace spočívala v přidání zápisového bufferu, který by se po naplnění teprve ukládal do souboru. Buffer by uchovávat přímo text, který bude zapsán.

Dále by bylo vhodné zápis na SD kartu provádět ve vlastním vlákně tak, aby nedocházelo k občasnému chvilkovému výpadku detekce kláves v důsledku momentálního zaneprázdnění zápisovou operací. Předávání kláves počítači by nemělo být ovlivněno, neboť se odehrává ve vlastním, k tomu určeném vlákně.

Nakonec by bylo dobré vyřešit alespoň částečně problém náhlého vysunutí SD karty. I když není možné přímo zajistit, že nedojde k poškození souborového systému, nemělo by dojít k zatuhnutí zařízení apod.

Řešení optimalizace

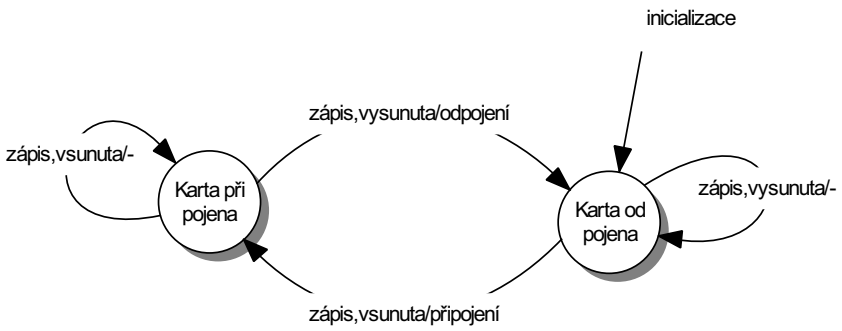
Výše zmíněné problémy bylo možné vyřešit přidáním dalšího ovladače, který je implementován v souborech `sd_write.c` a `sd_write.h`. Ovladač nevyužívá mechanismu device manageru, takže jej stačí pouze inicializovat. Inicializace je nutné provést před spuštěním plánovače systému VOS, jelikož ovladač využívá vláken.

Inicializace se provede zavoláním funkce `sd_write_init()`, které je předán odkaz na strukturu typu `sd_write_context_t`. Před ukončením činnosti ovladače nesmí dojít k jejímu přímému přepsání a porušení zapouzdření, či dealokaci. Ve struktuře je uchováván průběžný stav a nastavení ovladače.

Po spuštění plánovače je nutné ovladači předat ukazatel na handle ovladačů SD karty a FAT systému souborů zavoláním funkce `sd_write_attach_driver()`. Dalšími parametry jsou jméno souboru, které bude otevřeno pro zápis, a atributy, se kterými je potřeba soubor otevřít (zde `"w+"` pro doplnění na konec souboru). Oba poslední parametry se předávají jako ukazatel na řetězec. Na závěr je funkcí `sd_write_attach_card()` provedena inicializace připojené SD karty.

Následně je možné funkcí `sd_write()`, které je předán odkaz na řetězec znaků, zapisovat do zvoleného souboru. Při každém zavolání funkce je nejprve zkontrolována přítomnost karty. K tomu slouží funkce `sd_write_card_present()`, která pouze obaluje IOCTL volání předávané ovladači SD karty. Kontext ovladače udržuje předchozí stav karty, byla-li tedy karta připojena a je připojena stále, není třeba žádné další akce. Teprve při změně stavu je karta připojena, případně odpojena. Karta je tedy z ovladače SD odpojena i v případě, že se již ve slotu nenachází, to je však nutné pro uvolnění zdrojů a případné další připojení.

Obrázek konečného automatu 5.1 vystihuje chování ovladače při vyjmutí/vstunutí karty za běhu aplikace. Z obrázku je také patrné, že při zachování stavu vsunutí karty není prováděna další akce.



Obrázek 5.1: Mealyho konečný automat popisující přechody mezi stavy ovladače optimalizujícího zápis na SD kartu.

Následně je zkontrolován stav zápisového bufferu. Pokud by se do něj předávaná zpráva již nevešla, dojde k vyprázdnění bufferu a zápisu dat na SD kartu. Jde o překopírování dat do bufferu, vyhrazeného pouze pro zápisové vlákno a signalizaci zápisového vlákna `thread_write()`.

Výše uvedenými úpravami bylo dosáhnuto plynulé detekce stisku kláves. Nadále již nebyl zápis na kartu při běžném psaní postřehnutelný a nedocházelo ke ztrátám při detekci událostí. Zároveň však není doporučeno SD kartu ze zařízení vytahovat za běhu aplikace.

5.3.4 Formát ukládání záznamů

Záznamy jsou ukládány na konec souboru `log.txt` v kořenovém adresáři SD karty, takže nejstarší jsou uvedeny v souboru prvně. Formát ukládání stisků kláves je `KEYDOWN` *název_klávesy* pro událost stisku a `KEYUP` *název_klávesy* pro událost puštění klávesy. Tento způsob byl zvolen s ohledem na potřebu detekovat také současné držení několika kláves. Každý záznam události je uložen na vlastním řádku.

5.4 Zpřístupnění úložiště záznamů

Další vlastností zařízení by měla být možnost zpřístupnit úložiště záznamů po zadání hesla na připojené klávesnici.

Jako nejlepší a nejlépe podporované řešení mezi operačními systémy se nabízí vytvoření BOMS (Bulk-Only Mass Storage) zařízení. Takové zařízení by po zadání hesla zpřístupnilo vloženou kartu podobně, jako např. čtečka karet. Zařízení by se v operačním systému objevilo jako nový disk a bylo by možné s ním pracovat stejně jako např. s USB flashdiskem.

5.4.1 BOMS USB slave ovladač

Ovladač pro vytvoření USB slave zařízení třídy BOMS není standardně ve Vinculum II IDE dodáván, takže bude nutné jej navrhnout a implementovat. Ovladač s názvem BOMS, který je ve vývojovém prostředí dostupný, slouží pouze k připojení takového zařízení k obvodu VNC2, zařízení se poté chová jako USB host.

5.4.2 Návrh ovladače BOMS

Cílem je implementovat ovladač, který by převzal handle ovladače SD karty a ovladače USB slave portu. Oba tyto ovladače jsou standardně dodávány s Vinculum II IDE. Vzniklý ovladač by následně měl zpřístupnit úložiště SD karty pomocí protokolu bulk-only (popsaném v 2.6.2) jako USB slave zařízení hostiteli, do kterého je USB keylogger zapojený.

První dílčí fází byla implementace systému komunikace pomocí CBW a CSW (bulk-only protokol v mass storage podtřídě). Následně data získaná z pole CBWCB struktury CBW (viz 2.3) prohlásit za SCSI příkaz podle standardu SBC-2 [13].

Příkazová sada SCSI byla implementována jednotlivými funkcemi. Ty by byly volány podle operačního kódu uvedeného ve struktuře CBWCB. Stavová data by byla deklarována jako globální struktura, která by mohla být pozměňována jednotlivými funkcemi podle případné chyby. Po příkazu REQUEST SENSE (popsaném v 2.7.3) by došlo k odeslání této struktury.

5.4.3 Obecná struktura ovladače BOMS

Implementace BOMS je rozprostřena po souborech `usb_slave_boms.c`, `usb_slave_boms.h` a `usb_slave_boms_bw.h`. Ovladač využívá mechanismu device manageru. Ovladač při inicializaci vytvoří strukturu `usb_slave_boms_context_t` pro uchovávání aktuálního stavu. Po spuštění plánovače VOS je pro funkčnost využíváno tří vláken.

První vlákno slouží k obsluze standardních a třídních požadavků přijatých po sběrnici USB. Jedná se o ty požadavky, které využívají nultý endpoint. Po přijetí setup paketu je na základě jeho typu zpracováván funkcemi `boms_standard_request()` případně `boms_class_request()` podle postupu ovedeného v 4.1.2. Tyto se postarají o správné reakce na požadavky, které vznikají hlavně při enumeraci zařízení. Implementace standardních reakcí na setup pakety vychází ze zdrojového kódu ovladače HID klávesnice, který je dostupný po instalaci Vinculum II IDE a slouží mj. jako vzorový příklad.

Důležitou částí standardních požadavků je zasílání deskriptorů jako odpovědí na požadavky typu GET_DESCRIPTOR. Výše zmíněné zpracovává `boms_get_descriptor_request()`. Implementace je shodná s uvedený příkladem HID klávesnice, liší se však v obsahu deskriptorů. Kompletní výčet položek s jejich popisy je dostupný ve zdrojových kódech na začátku souboru `usb_slave_boms.c`.

Dalším vláknem je `usb_slave_boms_thread_in()`. Toto vlákno implementuje odesílání dat do hostitele po USB bulk endpointech. Odesílání ve vlastním vlákně dovoluje vytvořit operaci zápisu neblokující. Je využito základních mechanismů mezivláknové sychronizace, jako jsou mutexy (pro výlučný přístup k odesílacímu bufferu) a semafory (pro signalizaci nových dat).

Poslední vlákno `usb_slave_boms_thread_io()` implementuje zapouzdření příkazové sady SCSI do protokolu bulk-only.

5.4.4 Implementace bulk-only protokolu

Jedná se o čtení příchozích zpráv do bulk out endpointu (směr přenosu pojmenován ve vztahu k hostiteli) v nekonečné smyčce, jelikož čtení je blokující operací. Následně je ukazatel na data přijatá do bufferu uložen do kontextu ovladače. Tento způsob umožňuje pohodlné předávání velkého množství parametrů zároveň.

Následně je přijatý CBW zkontrolován funkcí `cbw_valid_and_meaningful()`. Funkce implementuje způsob kontroly validity a smysluplnosti bloku, popsaného ve specifikaci [17]. Při detekci chyby je příkazový blok zahozen a smyčka je přerušena - začíná čekání na nový blok dat. Implementaci usnadňuje fakt, že CBW má pevně danou délku 31B.

Po kontrole přichází zpracování přijaté položky CBWCB struktury CBW jako SCSI příkazu příkazové sady SBC-2. Zpracování zajišťuje funkce `process_scsi()`, která zároveň po interpretaci příkazu zajistí i zaslání odpovídající struktury CSW pomocí funkce `csw_response()`.

5.4.5 Problém s konvencemi pořadí bytů (endianness)

MCU zvolený pro realizaci používá konvence řazení bytů ve stylu little-endian. Takové řazení odpovídá ukládání méně významných bytů prvně (na nižší adresu). Až doposud nebylo potřeba tyto konvence uvažovat, protože podle zdrojů [32] a [17] ctí USB a protokol bulk-only stejné konvence.

Avšak SCSI podle [13] používá v příkazové sadě vícebytové parametry, a ty jsou uložené ve stylu big-endian, tedy nejvýznamnější byte na nejnižší adrese.

Pro vypořádání se s touto překážkou byly implementovány jednoduché funkce pro změnu konvencí řazení vícebytových hodnot pro datové typy `uint32` a `uint16`. Funkce se nazývají `switch_endianness_uint16()` a `switch_endianness_uint32()`, jejich jméno vyjadřuje typ, se kterým jsou schopny pracovat. Funkce přebírají jediný parametr, kterým je ukazatel na měněnou hodnotu. Po provedení je výsledek uložen na odkazovaném místě, kde byla původně zdrojová data.

5.4.6 Interpretace SCSI příkazů

Konstrukce `switch(CBWCB[0])` na základě prvního byte položky CBWCB (operačního kódu) rozhodne, o který příkaz se jedná. V praxi se často stává, že zařízení neimplementují každý příkaz, který je specifikací označen jako povinný, ale pouze jejich podmnožinu, která je nezbytná pro danou funkčnost [3]. V tomto případě bylo nutné implementovat následující příkazy.

READ(10)

Prvním zcela jistě nezbytným příkazem je READ(10). Po načtení parametrů a konverzi parametrů do konvence little-endian může být zahájen přenos dat.

V cyklu `for` jsou procházeny bloky od počátečního, daného hodnotou LBA, bloky jsou průběžně načítány z SD karty a odesílány pomocí neblokující funkce `usb_slave_boms_write()`.

Načítání z SD karty probíhá po blocích 512 bytů, bez účasti ovladače souborového systému. Způsob čtení po blocích je podrobně popsán v [7]. Jde v podstatě o využití struktury `_msi_xfer_cb_t`, které je potřeba dodat číslo bloku, délku čtených dat a další parametry přenosu. Struktura je poté přetypována na řetězec znaků a odeslána funkci `vos_dev_read()`.

WRITE(10)

Operace zápisu je implementačně téměř shodná s operací čtení, popsanou v 5.4.6. Rozdíl je pouze v použití volání `vos_dev_read()` pro čtení dat z bulk endpointů zařízení a `vos_dev_write()` pro zápis na SD kartu. Zápis využívá rovněž struktury `_msi_xfer_cb_t`.

INQUIRY

Odeslání informací o SCSI zařízení využívá předpřipravených informací, které jsou uloženy v `scsi_inquiry_response`, podobně, jako tomu je u USB deskriptorů.

Hostitel může požadovat pouze několik prvních bytů této informace, proto je nutné zohledňovat obsah `ctx->cbw->dCBWDataTransferLength`. Pokud je požadovaný obsah delší než `sizeof(scsi_inquiry_response)`, hostitel obdrží pouze dostupnou délku.

READ CAPACITY

Odeslání informace o kapacitě média využívá připravenou strukturu `usb_slave_boms_capacity_t`. Struktura má dvě položky `logical_block_address` a `block_len_in_bytes`.

Tyto položky je nutné naplnit údaji o kapacitě média. Dostupné informační zdroje použitého ovladače SD karty ovšem neobsahují zmínku, jak takovou informaci vyčíst. Proto je zde kapacita napevno dána definicí `LAST_BLOCK`. Omezení je popsáno v 5.4.8.

READ FORMAT CAPACITIES

Struktura je implementována pod názvem `usb_slave_boms_format_capacity_t`. Obsahuje informace o počtu bloků a délce jednoho bloku v bytech. Bližší popis je v 2.9.

Hodnota počtu bloků je dána hodnotou `LAST_BLOCK + 1` definovanou v souboru `usb_slave_boms.h`. Počet bytů v bloku je pevně dán na 512 bytů.

MODE SENSE

Příkaz MODE SENSE zapisuje nejmenší možnou strukturu podle popisu zmíněného v 2.5. Jde pouze o čtyři byty `0x03`, `0x00`, `0x00`, `0x00`.

REQUEST SENSE

Příkaz vrátí požadovaný počet tzv. sense dat. Ta jsou uložena jako struktura v paměti a jsou průběžně aktualizována podle stavu provedených příkazů (viz 5.4.2).

PREVENT/ALLOW MEDIA REMOVAL

Implementuje prázdný příkaz. Informace o možném vysunutí média nejsou pro tuto implementaci podstatné, jelikož zařízení neobsahuje prostředky pro zamezení vyjmutí karty.

TEST UNIT READY

Implementuje pouze reset sense dat do výchozích hodnot. SD karta je pamětí typu flash, tudíž je vždy signalizována připravenost média pomocí návratového kódu v CSW.

Neznámý příkaz

Pokud je přijat příkaz, který zařízení neimlementuje, je nutné o tom hostitele řádně informovat. To se provádí pomocí návratového kódu značícího chybu v příslušejícím CSW a příslušného nastavení sense dat.

Ve zdrojových kódech je použito definic `SCSI_STATUS_OK` a `SCSI_STATUS_FAIL`, které odpovídají návratovým kódům.

Že se jedná o neplatný příkaz následně upřesňuje nastavení položky sense key na hodnotu `5h`, ASC na `20h` a ASCQ na hodnotu `00h`. Struktura sense data je popsána v 2.6.

5.4.7 Mechanismus zpracování CSW

Jednotlivé SCSI příkazy ukládají do kontextu předaného ukazatelem jako parametr svoje chybové kódy. Proměnná obsahující stavový kód vždy poslední SCSI operace je pojmenována `last_cmd_status`. Tato hodnota je ve funkci `csw_response()` použita na místo položky bmCSWStatus struktry CSW, uvedené v 2.4.

5.4.8 Problémy a omezení implementace BOMS

Implementace ovladače BOMS obsahuje následující nedostatky.

Pevná velikost SD karet

Pro zpřístupnění SD karty byl vybrán ovladač firmy FTDI z důvodů jednoduché obsluhy a odladěnosti na vybraný HW. Implementovat vlastní ovladač SD karty na nejnižší úrovni by bylo možné, avšak vyžadovalo by to úsilí nepřiměřené nejistému výsledku. Dodávaný ovladač je navíc kompatibilní s ovladačem systému souborů FAT.

S ohledem na zmíněná fakta bylo omezení na pevnou velikost SD karet přijato. Z důvodů zajištění nejvyšší kompatibility mezi druhy karet byla výchozí hodnota velikosti nastavena na 512MB. Takové nastavení umožňuje použití všech karet, jejichž velikost je větší nebo rovna 512MB. Velikost úložného prostoru je tak ale omezena. Při nedostatku místa na SD kartě je možné změnit hodnotu `LAST_BLOCK` definovanou v souboru `usb_slave_boms.h` na nižší nebo rovnu velikosti používané SD karty.

Nemožnost vysunout kartu při běhu aplikace

SD karta nesmí být vyjmuta ze zařízení, pokud je zařízení zapnuté. Hrozí poškození systému souborů na kartě a ztráta uložených dat. Důvodem je nemožnost předpovědět vysunutí karty tak, aby případně probíhající operace zápisu zanechaly systém v konzistentním stavu.

Požadavky na rychlost zdroje dat

Při testování a ladění ovladače BOMS docházelo k potížím při přenosu větších souborů (v řádu 1MB a více). Komunikace problíhala standardním způsobem do chvíle, než hostitel přestal vyzvedávat obsah endpointů pro datovou výměnu.

Domnělým důvodem tohoto chování je nízká rychlost při vyřizování datových operací a timeout na straně hostitele. Při pokusu odesílat v datových blocích konstantní hodnoty odvozené od adresy bloku LBA k těmto problémům nedocházelo. Pokud tedy zařízení data nestahovalo z SD karty, což způsobuje jisté prodlevy, probíhala i delší komunikace bez problémů. Tento problém se nepodařilo vyřešit a tak je možné přenášet pouze menší soubory.

Jedná se tedy pravděpodobně o důsledek využití již existujícího ovladače SD, který zde způsobuje úzké hrdlo.

5.5 Návrh šifrování pro úložiště

Vzhledem k problémům, které nastaly při implementaci ovladače BOMS (5.4.8) nebylo šifrování implementováno v konečném řešení. Následuje tedy pouze návrh knihovny, která nebyla do výsledku zařazena z důvodu již tak nízké propustnosti, zjištěné měřením 5.7.

5.5.1 Rozhraní a návrh implementace

Aby bylo možné ovladač pro šifrování připojit transparentně mezi ovladač FAT a ovladač SD, je nutné zachovat rozhraní MSI (Mass Storage Interface), které používají ovladače firmy FTDI. Tak je zajištěna kompatibilita.

Implementace by spočívala ve vytvoření obálky pro ovladač podporující MSI. Pomocí IOCTL volání by byl šifrovacímu ovladači předán handle na instanci ovladače MSI. Šifrovací ovladač by všechny požadavky předával připojenému ovladači MSI beze změny, kromě dat. Ta by podle směru přenosu zašifroval či dešifroval. V níže zmíněné variantě však není směr přenosu nutné brát v úvahu.

5.5.2 Šifrovací algoritmus

Následující podsekce čerpá z [35].

Šifrování by probíhalo pomocí pevného klíče. V bloku by se procházely sekvence bitů po násobcích délky klíče a pro každou takovou sekvenci by se provedla operace xor s daným klíčem.

U zmíněného algoritmu je možné poměrně jednoduše odhalit šifrovací klíč, avšak nemožnost přečíst data na kartě pouhým vložením do čtečky karet bez dalších akcí by byla splněna. Algoritmus se ale spíše hodí pro skrývání informací než jejich zabezpečení.

5.5.3 Implementace šifrování

Ovladač je implementován v souborech `encrypt.c` a `encrypt.h` v adresáři `/encrypt` na přiloženém CD.

5.6 Přepínání mezi režimy keylogger a čtečka SD

K přepnutí režimu ze záznamů kláves do režimu čtečky SD karet je nutný restart MCU. Nastává ovšem problém uchování stavu. Obvod po restartu naběhne opět do stejného režimu.

Z dostupných zdrojů není patrné, jakým způsobem je možné využít vnitřní flash paměť zvoleného obvodu VNC2 z kódu programu, aby do ní bylo možné ukládat data perzistentně tak, že zůstanou zachována i po restartu obvodu. Flash paměť v obvodu bezpochyby přítomna je.

5.6.1 Návrh jednobitové paměti

Z výše uvedeného důvodu bylo nutné vytvořit HW prostředek, který by umožňoval krátkodobé zapamatování jednobitové hodnoty. Řešením bylo připojit na volný pin obvodu VNC2 kondenzátor, který by byl schopen si dočasně pamatovat stav.

5.6.2 Uložení a načtení stavu při přepínání režimu

K uložení hodnoty dochází po detekci správného hesla. Při stisku klávesy, která přísluší písmenu, je toto písmeno uloženo do staticky alokované proměnné ve funkci `password()`, která implementuje kruhový buffer. Ten při shodě s heslem definovaným v souboru `password.h` pod jménem `PASS` vyvolá nastavení pinu č. 12 jako výstupního a zápis logické jedičky na tento pin, po 100ms následuje restart obvodu. Pin 12 je součástí portu A. Výchozí heslo zní `testpass`.

Ve vstupním bodu programu, funkci `main()`, je po spuštění funkce `vos_gpio_set_port_mode()` nastaven `GPIO_PORT_A` do režimu čtení a pomocí `vos_gpio_read_port()` jsou načteny do proměnné `mode` hodnoty pinů portu A. Následně jsou pomocí masky a logické operace AND odstraněny nezajímavé bity a ponechán pouze pin 12 (IOBUS1). Tím je vyčten stav nabití kondenzátoru C15, který uchovává budoucí režim právě spuštěné aplikace. Následuje jednoduché větvení inicializace podle vyčeného stavu.

Pro vyšší přehlednost jsou inicializace jednotlivých režimů (záznam kláves a čtení SD karty) rozděleny do příslušných funkcí `keylogging_main()` a `cardreading_main()`. Jednotlivé větve programu nadále pokračují v souborech `threads_keylogging.c` a `threads_keylogging.h`. Ve funkci `main()` poté dochází pouze ke spuštění plánovače VOS.

Drobným omezením celé implementace je nedostupnost klávesnice, po uvedení zařízení do režimu SD čtečky. Pro opětovné spuštění zařízení v režimu záznamu kláves je nutné ho odpojit a znovu připojit k hostiteli. Pro instanci ovladačů klávesnic už nezbyla volná kapacita v paměťi RAM.

5.7 Měření propustnosti výsledného řešení

Cílem měření je identifikovat úzké hrdlo při použití ovladače BOMS slave z uvedeného řešení. Celkem byly navrženy čtyři testovací firmware. Testovací firmware byly přeloženy v prostředí Vinculum II IDE s nastavením `Release` a postupně nahrány do MCU. Na hostitelském počítači byl spuštěn skript, který byl navržen pro účel tohoto měření.

Skript pro odesílání bloků 512B dat byl napsán v jazyce Python 2.6 za pomoci knihovny PyUSB, která je k dispozici v repozitáři distribuce Debian 6 pod názvem `python-usb`. Testování se odehrávalo na osobním počítači s procesorem Pentium 4, USB 2.0 v nezatíženém

stavu, na operačním systému Debian Linux 6.0.7 s Python 2.6.6. Klientská část aplikace je rovněž dostupná na přiloženém CD v adresáři `/throughput/client`.

Tabulka 5.1 uvádí výsledky měření propustnosti uvedeného řešení. Pro srovnání byla měřena také propustnost pouze částečného zpracování paketů a zahazování paketů (pouze příjem). Firmware pro tento účel je dostupný z přiloženého CD v adresáři `/throughput/firmware`.

Údaje z tabulky 5.7 vztahující se k použití šifrování XOR, jsou pro zasílání 512B bloků dat do MCU a spouštění algoritmu popsaného v 5.5.2 nad těmito daty. Test rychlosti SD je při příjmu 512B dat po USB a následném blokujícím zapise na kartu. Využívá ovladače firmy FTDI pro SD kartu a SPI řadič. Údaje pro šifrování a zápis jsou kombinací dvou výše zmíněných. Zápis na kartu se provádí na bloky adresované od nuly, jdoucí za sebou tak, aby nedošlo k poškození karty přílišným množstvím zápisů na stejnou adresu.

Poslední hodnoty týkající se hotového řešení USB BOMS slave ovladače byly jednoduše získány načtením 128 bloků 512B dat programem `dd`, který je běžně dostupný v Linuxových distribucích. Aby bylo možné vyloučit vliv vyrovnávací paměti na straně OS počítače, byly bloky načítány vždy od jiné adresy tak, aby průniky rozsahů adres načítaných bloků byly prázdné. Jednalo se konkrétně o příkaz
`dd if=/dev/sd`*X* `of=/dev/null ibs=512 count=128 skip=`*počáteční_blok*.

Z výsledků lze usoudit, že úzkým hrdlem navrženého řešení je právě ovladač SD karty ve spojení s pomalým zpracováním dat.

Zpracování dat	Doba přenosu 4096 * 512B bloků [s]				Výsl. propustnost [KiB/s]
	Měření 1	Měření 2	Měření 3	Průměr	
Zahazování paketů	4,094717s	4,100756s	4,097697s	4,0977233s	499,7897KiB/s

Zpracování dat	Doba přenosu 128 * 512B bloků [s]				Výsl. propustnost [KiB/s]
	Měření 1	Měření 2	Měření 3	Průměr	
Šifrování bloků operací XOR	6,767809s	7,080754s	6,7228s	6,857121s	9,3334KiB/s
Zápis bloků za sebe na SD	15,9329289s	15,848856s	15,690078s	15,8239543s	4,0445KiB/s
Šifrování a zápis bloků za sebe na SD	23,849538s	24,03262s	24,084347s	23,988835s	2,6679KiB/s
Čtení z SD, BOMS ovladač	68,1511s	68,4131s	68,3021s	68,2887667s	0,9372KiB/s

Tabulka 5.1: Výsledky měření propustnosti USB slave portu při zpracování dat na MCU.

Kapitola 6

Hardwarová realizace

Pro tvorbu schématu zapojení bylo využito programu Eagle verze 6.4.0, který je zdarma v omezené verzi dostupný z [5]. Pro následující tvorbu desky plošných spojů je však omezená verze zcela dostačující.

6.1 Tvorba schématu zapojení

Schéma je dostupné v přiloze D a také na CD v adresáři `/board/eagle`.

Program Eagle je dodáván s množinou knihoven, které obsahují nepřeberné množství elektronických součástek. Přesto však neobsahovaly všechny potřebné.

Pro určení, ve kterém režimu se zařízení spustí slouží kondenzátor C15. Ten je před restartem čipu nabit na hodnotu napětí 3,3V. Kondenzátor je zapojen paralelně s odporem R12, 47kOhm, který zajišťuje postupné vybíjení a hlavně propojuje pin obvodu VNC2 se zemnícím pólem. Tak je zajištěna správná detekce, pokud je kondenzátor vybitý.

Součástky z externích Eagle knihoven

Jedná se zejména o použitý MCU VNC-2-32L1B, pro který ovšem existuje již hotová knihovna dostupná v [34] pod jménem `ftdichip-5.lbr`. Výše zmíněný typ pouzdra součástky byl zvolen pro snažší zapájitelnost do DPS.

Další součástka z externích knihoven je feritová perla rozměrů 0805. Knihovna obsahující tuto součástku je dostupná z [30].

Knihovna PaJa je rozsáhlá. Zde byla použita kvůli přehledně uváděným roztečím vývodů u diskrétních součástek. Konkrétně u elektrolytického kondenzátoru C15. Knihovna je dostupná z [33] pod názvem `PaJa_30`.

Další součástkou, nedostupnou v dodávaných knihovnách, byl programovací konektor označený jako `DEBUG_IF`, který má méně obvyklou rozteč pinů 2mm a šest pinů. Součástka byla vytvořena v editoru součátek Eagle a byla uložena do knihovny `2mm_header.lbr`. Je dostupná na přiloženém CD v adresáři `/board/eagle_libs`.

Poslední součástkou je držáček microSD označený jako DM3D-SF od firmy HIROSE. Ten bylo nutné rovněž vytvořit a je přiložen v knihovně `microsd_hirose.lbr` na přiloženém CD. Tato a výše zmíněná knihovna obsahují každá pouze jednu uvedenou součástku.

6.2 Tvorba rozložení na DPS

Při návrhu rozložení součástek bylo nutné dbát zejména na datové vodiče USB. Ty je vhodné nekřížit přes vrstvy DPS a zároveň by příslušné vodiče měly mít přibližně stejnou délku.

Z důvodů minimalizace DPS byl držáček microSD karty umístěn na opačnou stranu od ostatních součástek.

Krystal Q1 a pomocné C9 a C8 byly umístěny co nejblíže MCU. Poté bylo využito funkce `Autorouter` a případné nedostatky byly ručně opraveny. Při tvorbě návrhu DPS byla využita funkce `Polygon` programu Eagle, která umožňuje automatické rozlití plošek `GND` po volných místech desky.

Maska DPS je součástí přílohy C.

Kapitola 7

Závěr

Cílem bylo implementovat prototyp zařízení USB keylogger, které zaznamenává stisky kláves připojené klávesnice. Zároveň po stisku určité kombinace kláves se zpřístupní úložitě těchto záznamů. Úložiště mělo být šifrováno.

Prvního cíle bylo po zvážení dosaženo s využitím dostupných knihoven firmy FTDI. Zpřístupnění úložitě bylo nutné implementovat na úrovni výměny zpráv po sběrnici USB. I přes značné úsilí má však uvedené řešení ovladače nedostatky zejména z hlediska rychlosti přenosu souborů do/z úložitě. Jde pravděpodobně o chybu způsobenou na straně ovladače SD firmy FTDI. Ovladač podporuje pouze velikost karet uvedenou ve zdrojových kódech, což je způsobeno nemožností vyčíst velikost připojené karty ovladačem SD. Zejména tyto nedostatky by bylo nutné odstranit v případném dalším vývoji této práce.

Již tak nízká rychlost znemožnila prakticky implementovat šifrování úložiště, a tak bylo řešení implementováno pouze jako modul, který není do výsledného řešení zahrnut.

Práce tedy splnila všechny cíle, kromě uvedeného šifrování úložiště.

Zajímavým rozšířením by mohlo být nastavování zařízení pomocí konfiguračních souborů, které by se ukládaly na SD kartu, včetně možnosti změny rozložení kláves na klávesnici a snazšího update firmware.

Hlavím přínosem řešeného úkolu je implementace vlastního ovladače BOMS slave, který lze po drobných úpravách využít i v jiných případech, než je jen zpřístupnění SD karty. Pro mě byla přínosem zkušenost z tvorby rozsáhlejší vestavěné aplikace pro USB a zkušenost při tvorbě vícevrstvé DPS.

Fotografie hotového zařízení jsou k dispozici v příloze E.

Literatura

[1] Axelson, J.: *USB Complete, Third Edition, Everything You Need to Develop Custom USB Peripherals*. Lakeview Research LLC, 2005, iSBN 1931448027.

[2] Axelson, J.: *USB Mass Storage, Designing and Programming Devices and Embedded Hosts*. Lakeview Research LLC, 2006, iSBN10 1-931448-05-1.

[3] Axelson, J.: The Mass Storage Page. http://www.lvr.com/mass_storage.htm, [cit. 2013-05-01].

[4] Bidlo, M.: Synchronní sériová rozhraní SPI, I2C [online]. https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IMP-IT/lectures/P4/SeriovyPrenos_SPI_IIC.pdf?cid=8661, [cit. 2012-11-28].

[5] CadSoft: Download latest version of Eagle [online]. http://www.cadsoftusa.com/download-eagle/, [cit. 2013-05-01].

[6] Freescale Semiconductor, Inc.: M68HC11RM Datasheet. http://www.freescale.com/files/microcontrollers/doc/ref_manual/M68HC11RM.pdf, [cit. 2012-12-26].

[7] FTDI Ltd.: Vinculum II User Guide. http://www.ftdichip.com/Support/Documents/AppNotes/AN_151_Vinculum-II_User_Guide.pdf, [cit. 2012-01-21].

[8] FTDI Ltd.: V2-EVAL Datasheet. http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_V2-EVAL.pdf, [cit. 2013-04-21].

[9] FTDI Ltd.: Vinculum II. http://www.ftdichip.com/Products/ICs/VNC2.htm, [cit. 2013-04-21].

[10] FTDI Ltd.: Vinculum II Development Tools. http://www.ftdichip.com/Firmware/VNC2tools.htm, [cit. 2013-04-21].

[11] Kotásek, Z.: Charakteristika rozhraní USB [online]. https://www.fit.vutbr.cz/study/courses/IPZ/public/texty/rok2010/usb10/usb2010.pdf, [cit. 2012-11-21].

[12] SD Group: SD Specification, Part1, Physical Layer, ver. 3.01 [online]. https://www.sdcard.org/downloads/pls/simplified_specs/Part_1_Physical_Layer_Simplified_Specification_Ver_3.01_Final_100518.pdf, 2010-05-18 [cit. 2012-11-28].

[13] Seagate: SCSI Commands Reference Manual, Rev. A. http://www.seagate.com/staticfiles/support/disc/manuals/scsi/100293068a.pdf, [cit. 2013-05-01].

[14] Straka, M.; Šimek, V.; Kaštil, J.: Analýza komunikace na sběrnici USB [online]. https://www.fit.vutbr.cz/study/courses/IPZ/public/laboratore/rok2008/ipz-usb-prez.pdf, 2008 [cit. 2012-11-21].

[15] Strnadel, J.: Vestavěné aplikace - Případové studie [online]. https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IMP-IT/lectures/P9-2012/imp_embeddedCasesSDFAT.pdf?cid=8661, [cit. 2012-11-28].

[16] T10 Technical Commitee: SCSI Multimedia Commands - 2 (MMC-2), Rev. 11a [online]. http://www.rockbox.org/wiki/pub/Main/DataSheets/mmc2r11a.pdf, 1999-08-30 [cit. 2013-05-01].

[17] USB Implementers' Forum: Universal Serial Bus, Mass Storage Class, Bulk-Only Transport [online]. http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf, 1999-09-31 [cit. 2013-05-01].

[18] USB Implementers' Forum: Device Class Definition for Human Interface Devices (HID) [online]. http://www.usb.org/developers/devclass_docs/HID1_11.pdf, 2001-06-27 [cit. 2013-05-01].

[19] WWW stránky: A Series of Articles on USB [online]. http://www.usbmadesimple.co.uk/index.html, [cit. 2012-11-20].

[20] WWW stránky: Universal Serial Bus [online]. http://en.wikipedia.org/wiki/Universal_Serial_Bus, [cit. 2012-11-20].

[21] WWW stránky: Stop and Wait ARQ [online]. http://en.wikipedia.org/wiki/Stop-and-wait_ARQ, [cit. 2012-11-21].

[22] WWW stránky: USB 3.0 Interface Bus, Cable Diagram [online]. http://www.interfacebus.com/usb-cable-diagram-30.html, [cit. 2012-11-21].

[23] WWW stránky: USB 3.0 [online]. http://en.wikipedia.org/wiki/USB_3.0, [cit. 2012-11-21].

[24] WWW stránky: USB.org [online]. http://www.usb.org/home, [cit. 2012-11-21].

[25] WWW stránky: SD Cards, SDHC [online]. http://www.centerpointaudio.com/SDCard.aspx, [cit. 2012-11-28].

[26] WWW stránky: Secure Digital [online]. http://en.wikipedia.org/wiki/Secure_Digital, [cit. 2012-11-28].

[27] WWW stránky: Serial Peripheral Interface Bus [online]. http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus, [cit. 2012-11-28].

[28] WWW stránky: Beyond Logic, USB in a NutShell [online]. http://www.beyondlogic.org/usbnutshell/, [cit. 2013-02-11].

[29] WWW stránky: Keyboard Scancodes: USB. http://www.win.tue.nl/~aeb/linux/kbd/scancodes-14.html, [cit. 2013-04-21].

[30] WWW stránky: AWJ Logan Public Eagle Library [online]. http://awjlogan.wordpress.com/2012/06/25/public-eagle-library-released/, [cit. 2013-04-24].

[31] WWW stránky: Elastic Sheep, Reading an SD card with an ATMEGA168 [online]. http://elasticsheep.com/2010/01/reading-an-sd-card-with-an-atmega168/, [cit. 2013-04-24].

[32] WWW stránky: Endianness [online]. http://en.wikipedia.org/wiki/Endianness, [cit. 2013-04-24].

[33] WWW stránky: Paja, Eagle [online]. http://paja-trb.cz/eagle/, [cit. 2013-04-24].

[34] WWW stránky: Userfiles, libraries [online]. http://web.cadsoft.de/cgi-bin/download.pl?page=/home/cadsoft/html_public/demo.htm&dir=eagle/userfiles/libraries, [cit. 2013-04-24].

[35] WWW stránky: XOR cipher [online]. http://en.wikipedia.org/wiki/XOR_cipher, [cit. 2013-05-05].

Příloha A

Použité zkratky

BOMS Bulk-only Mass Storage

CBW Command Block Wrapper

CRC Cyclic Redundany Check

CSW Command Status Wrapper

DPS Deska plošných spojů

FAT File Allocation Table

FW Firmware

HID Human Input Device

HW Hardware

MCU Microcontroller Unit

MISO Master In Slave Out

MOSI Master Out Slave In

MSI Mass Storage Interface

OS Operační systém

PCB Printed Circuit Board

SCSI Small Computer Systems Interface

SD Secure Digital

SPC SCSI Primary Commands

SPI Serial Peripheral Interface

SS Slave Select

SW Software

USB Universal Serial Bus

VOS Vinculum Operating System

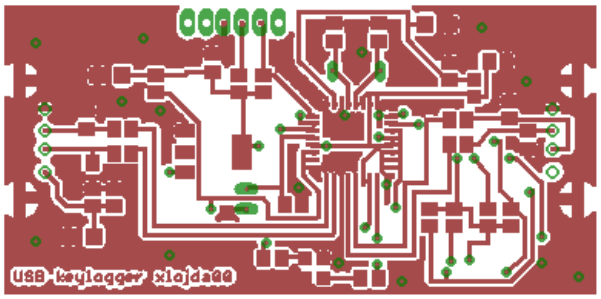
Příloha B

Obsah CD

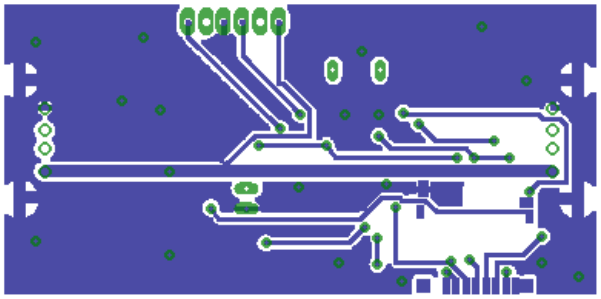
CD ROOT	
|- board	
| |- eagle	Soubory Eagle projektu se schématem a rozložením DPS
| |	
| |- eagle_libs	
| | |- 2mm_header.lbr	Knihovna obsahující pin header s 2mm roztečí
| | |- microsd_hirose.lbr	Knihovna obsahující držáček microSD (DM3D-SF)
| |	
| |- pool	Konvertované soubory pro výrobu DPS
|	
|- encrypt	Ovladač pro šifrování úložiště s rozhraním MSI
|	
|- text	
| |- bp.pdf	Práce v PDF formátu
| |- src	Zdrojový text práce
|	
|- throughput	
| |- client	Implementace klientské části k měření propustnosti USB
| |	
| |- fw	Implementace FW zahazující pakety pro
|	účely měření propustnosti, projekt pro Vinculum II IDE
|	
|- usbkeylogger	Implementace FW keyloggeru, projekt pro
	Vinculum II IDE

Příloha C

Rozložení součástek na DPS



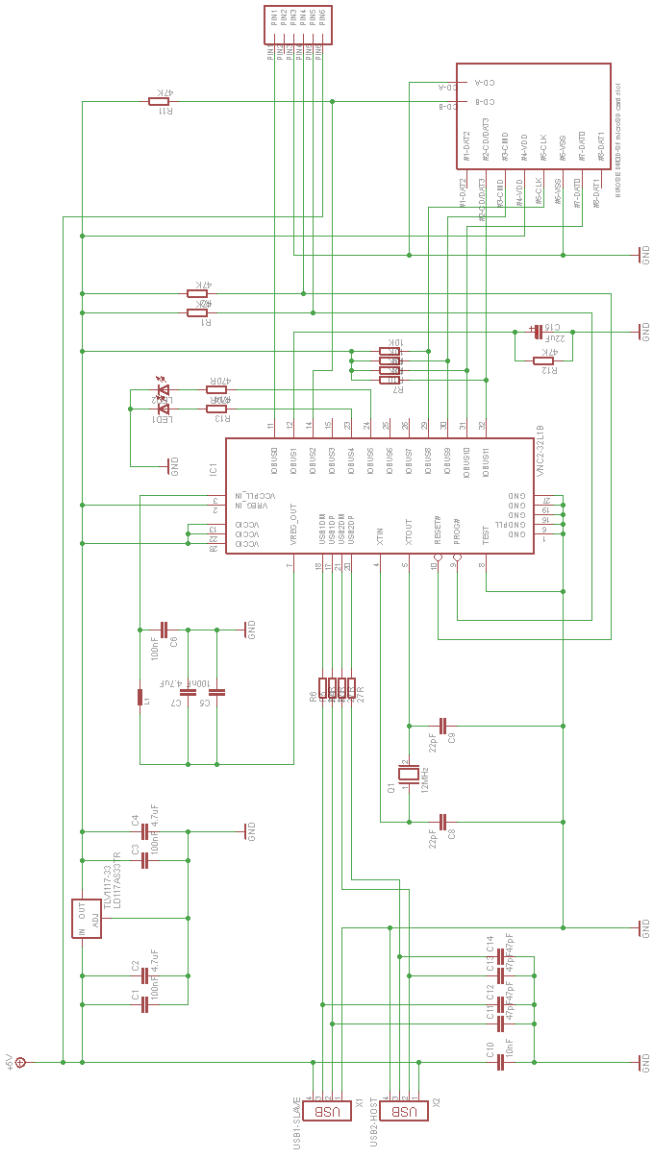
Obrázek C.1: Horní vrstva plošného spoje



Obrázek C.2: Spodní vrstva plošného spoje

Příloha D

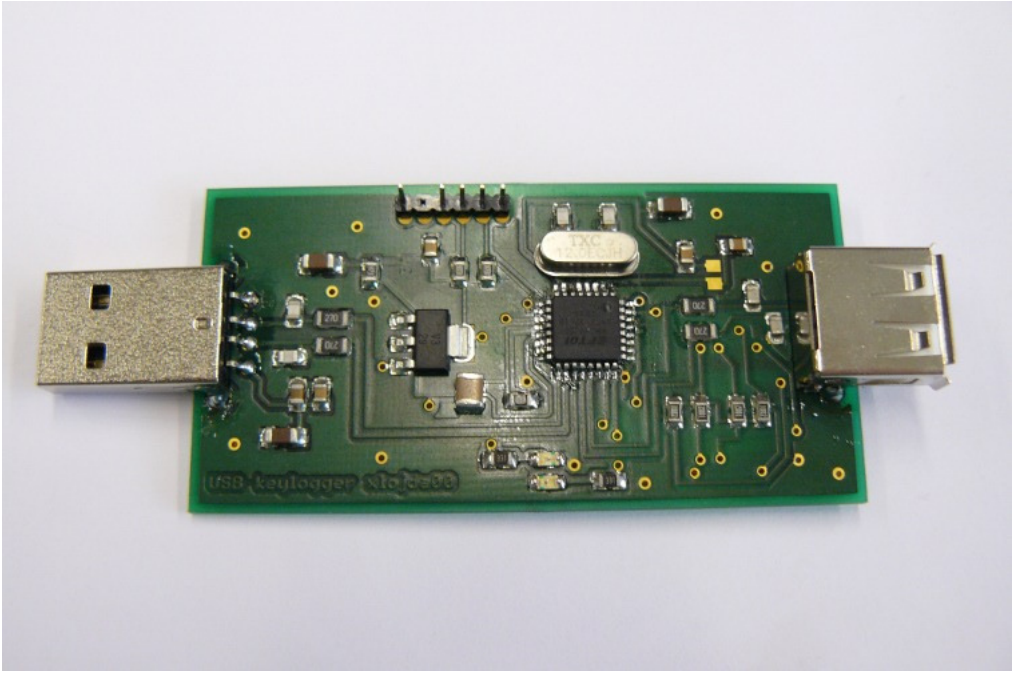
Schéma zapojení



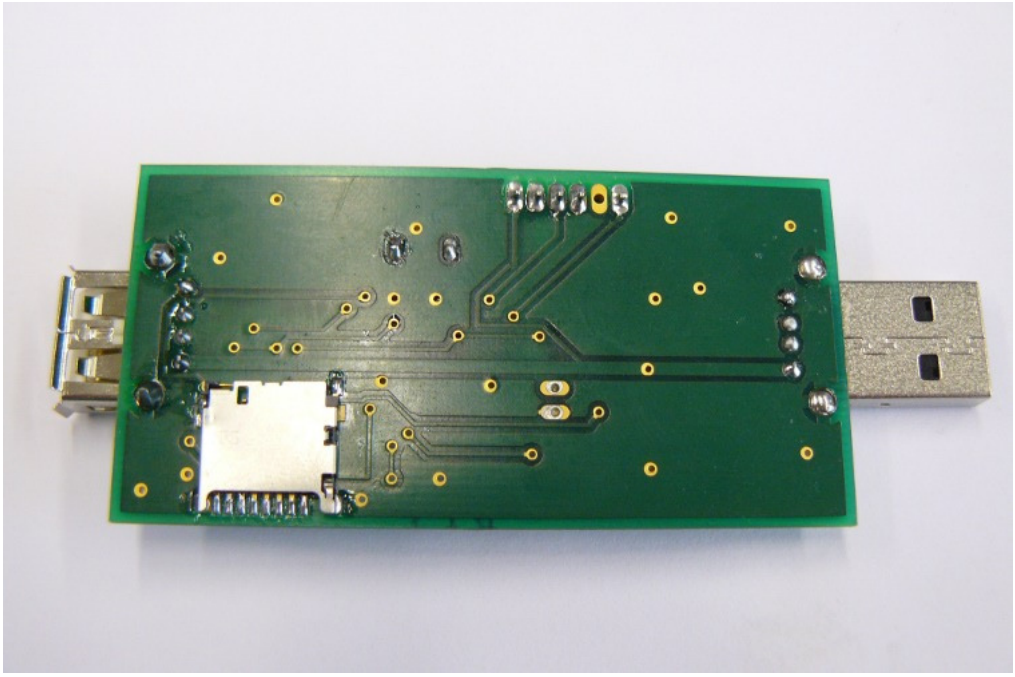
Obrázek D.1: Schéma zapojení

Příloha E

Fotografie zařízení



Obrázek E.1: Fotografie hotového zařízení, pohled ze strany většiny součástek.



Obrázek E.2: Fotografie hotového zařízení, pohled ze strany se slotem na SD kartu.

Příloha F

Seznam součástek

Kondenzátory	
4x	100nF smd 0805, 16V
2x	22pF smd 0805, 16V
4x	47pF smd 0805, 16V
1x	10nF smd 0805, 16V
1x	20uF elektrolytický, rozteč 2mm, 16V
3x	4,7uF smd 1206, 16V
Rezistory	
2x	470R, smd 0805
4x	27R, smd 0805
4x	47k, smd 0805
4x	10k, smd 0805
Konektory a držáček	
1x	USB A konektor samice
1x	USB A konektor samec
1x	D3DSF Hirose držácek microSD
1x	Pin header, rozteč pinů 2mm
Ostatní	
1x	Ferrite bead 0,1R @ 600MHz, smd 0805
2x	LED zelená, smd 0805
1x	Krystal 12Mhz, rozteč pinů 5mm
1x	Stabilizátor napětí TLV1117-3,3V
1x	MCI Vinculum VNC2, 32 pinů, LQFP pouzdro

Tabulka F.1: Seznam potřebných součástek.