České vysoké učení technické v Praze
Fakulta elektrotechnická

DIPLOMOVÁ PRÁCE

Vývoj zařízení PROFINET IO Device

Karel Boček

Borek, 2009

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

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

V Borku dne _____ _____

podpis

Poděkování

Děkuji zejména vedoucímu diplomové práce Pavlu Burgetovi za přípravu tématu diplomové práce a rady, které pomohly při jeho vypracování. Dále děkuji všem na galerce, především Martinovi Samkovi a Pavlu Mezerovi, za trpělivost a čas, který mi věnovali při vývoji ovladače a zprovoznění desky SHARK.

Abstrakt

Cílem této práce je vytvořit komunikační periferii umožňující komunikaci po protokolu PROFINET. Zařízení pracuje jako PROFINET IO device. Při vývoji zařízení byly použity open-source nástroje v maximálním možném rozsahu.

Protokol PROFINET je nástupcem velmi rozšířeného protokolu PROFIBUS, proto od něj lze očekávat široké nasazení. Jako fyzickou vrstvu používá klasický ethernet a řadí se do rodiny průmyslového ethernetu s možností komunikace v reálném čase. Zde popsané řešení splňuje požadavky pro PROFINET IO RT třídu 1.

Abstract

The goal of this work is to prepare a communication periphery unit enabling to communicate as a PROFINET IO Device. Open-source software has been used for development as much as possible.

The PROFINET protocol as a successor of the well-known PROFIBUS fieldbus has high expectations as one of the industrial-Ethernet protocols, being able to be used for real-time communication. The presented solution conforms to the RT Class 1 communication characteristics of the PROFINET IO protocol.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Karel Boček**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Vývoj zařízení Profinet IO Device**

Pokyny pro vypracování:

1. Seznamte se s komunikací Profinet IO RT class 1.
2. Realizujte zařízení Profinet IO device s čipem Hilscher NetX na platformě PC a operačním systému Linux.
3. Realizovaný software přeneste na platformu s PowerPC a procesorovou deskou Shark s rozhraním pro čip NetX s operačními systémy Linux a PikeOS.
4. Porovnejte aplikaci Profinet IO Device pro oba operační systémy vzhledem k době odezvy komunikace na síti a komunikace s nadřazenou aplikací.
5. Otestujte realizované zařízení v síti Profinet IO spolu s dalšími komerčně dostupnými zařízeními a vypracujte testovací protokol. Testování proveďte jako podmnožinu testovacích případů podle specifikací organizace Profibus International.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Pavel Burget, Ph.D.

Platnost zadání: do konce letního semestru 2009/10

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 27. 2. 2009

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Cílem této práce je vytvořit komunikační periferii pro vestavná zařízení, která bude umožňovat komunikaci po protokolu PROFINET IO RT Class1. Dále musí řešit zpracování stacku PROFINETu a co nejméně zatěžovat host CPU.

Zvolené řešení je založené na specializovaném komunikačním čipu netX německé firmy Hilscher. Tento čip je specializovaný na průmyslovou komunikaci a splňuje všechny výše uvedené požadavky.

Jako host CPU bude použito běžné PC s OS Linux pro přípravu. Cílovou platformou bude deska SHARK s procesorem Freescale MPC5200. V tomto zařízení poběží OS Linux nebo PikeOS.

Autorem protokolu PROFINET je německá firma Siemens, která je předním výrobce automatizační techniky. Protokol PROFINET je nástupce velmi rozšířeného protokolu PROFIBUS a očekává se od něj obdobný úspěch. Liší se zejména použitou fyzickou vrstvou, PROFINET používá běžné kabely a síťové prvky ethernetu. Patří mezi tzv. průmyslové real-time ethernetové protokoly. Tyto protokoly se začínají v současné době rozšiřovat zejména díky možnosti využití stávajících síťových rozvodů a možnosti napojení na nadřazené systémy řízení výroby (MES a ERP).

V této práci bude postupně popsána komunikace po PROFINETu, dále ovladač pro čip netX a nakonec testování celého zařízení.

Kapitola 2

PROFINET RT

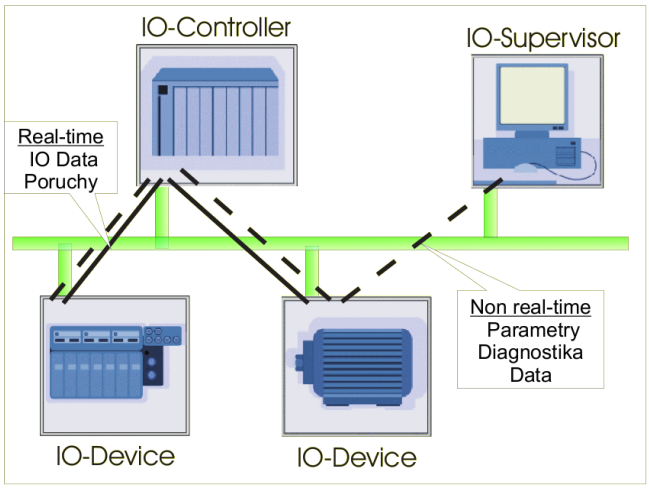
2.1 Popis komunikačního protokolu PROFINET IO RT Class1

Popis zde uvedený si nedává za cíl kompletní popis komunikačního protokolu PROFINETu, toto je již obsaženo v literatuře (viz [1]) a jiných pracích (viz [2] a [3]).

2.1.0.1 Fyzická vrstva a topologie

Protokol PROFINET IO RT používá jako fyzickou vrstvu běžné rozvody ethernetu, nevyžaduje stínění síťových kabelů.

Topologie zapojení je na obrázku 2.1.

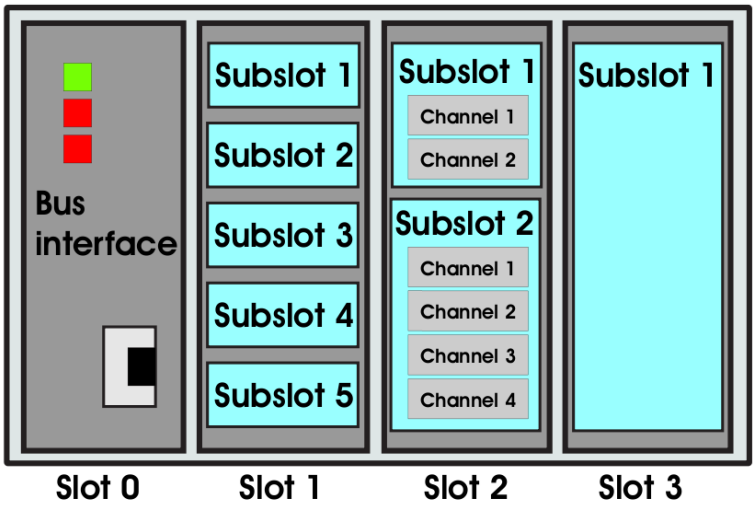


Obrázek 2.1: Topologie zařízení pro PROFINET (zdroj: [2])

PROFINET IO controller je zařízení, které vykonává požadovaný řídící program (typicky PLC). **PROFINET IO device** je zařízení, které slouží jako vzdálená inteligentní periférie. Oba dva tyto termíny se nepřekládají a jsou široce používány ve následujícím textu.

2.1.0.2 Struktura zařízení IO device

Pro vývoj zařízení PROFINET IO device je třeba seznámit se s jeho strukturou z pohledu protokolu PROFINET. Zařízení je koncipováno jako modulární a umožňuje vysokou flexibilitu. Základní struktura je na obrázku 2.2.



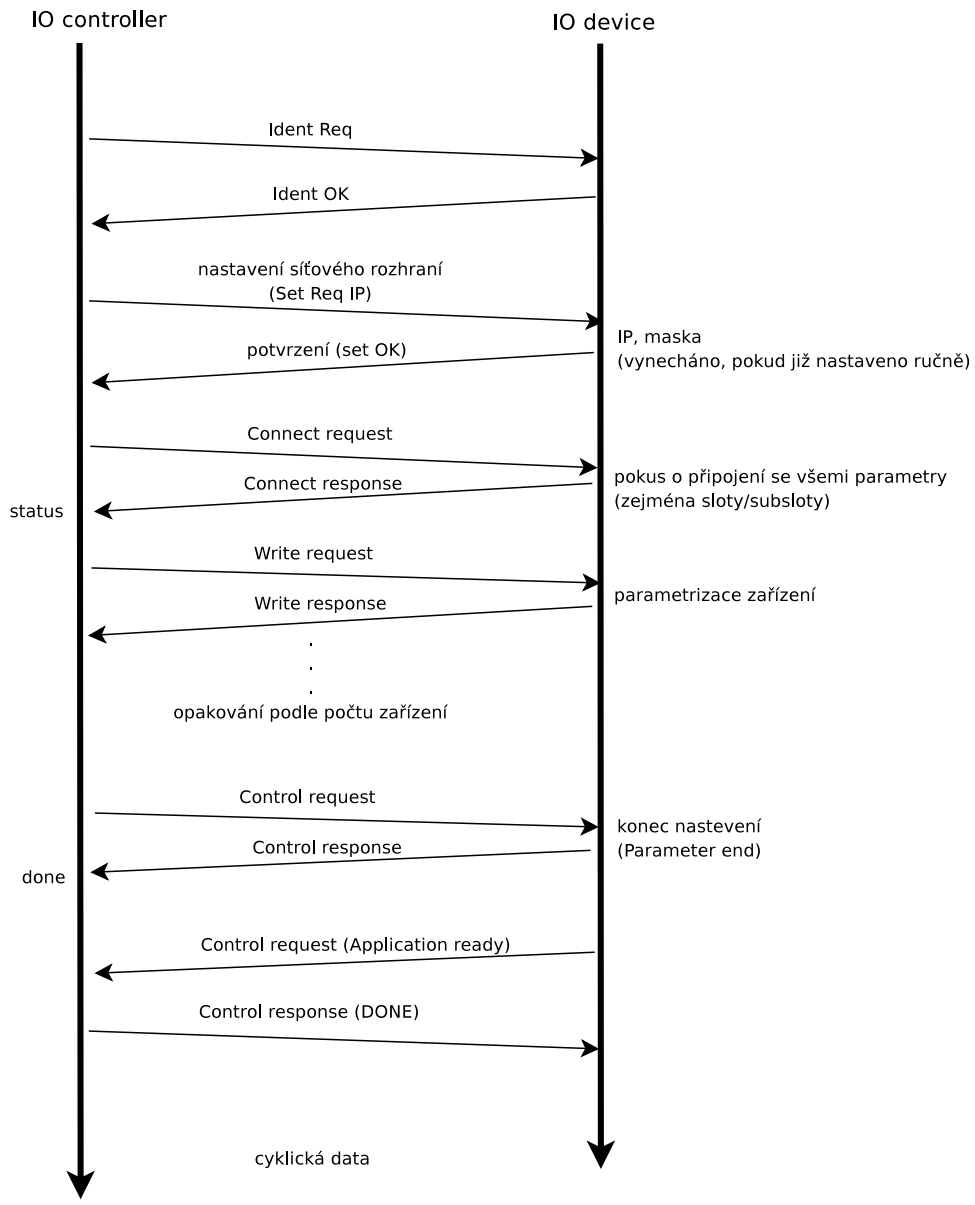
Obrázek 2.2: Struktura zařízení IO device

Kde jednotlivé subsloty reprezentují fyzické vstupy/výstupy, sloty jsou pro ně nadřazenou skupinou. Díky této struktuře lze věrně popsat fyzickou strukturu zařízení. Slot 0 je systémový a neslouží pro předávání cyklických dat, slouží jako síťové rozhraní.

2.1.1 Navázání spojení controller-device

Komunikaci zahajuje vždy controller, zařízeni IO device nikdy nesmí zahájit komunikaci. Proto pro potřeby testování zařízení PROFINET IO device je vždy nutné mít připojen controller.

Průběh navázání komunikace je podrobně popsán na obrázku 2.3.



Obrázek 2.3: Průběh navázání spojení controller - device

Prvním paketem je vždy tzv. Ident request (požadavek identifikace viz [1]). Tímto paketem controller hledá zařízení IO device, v paketu je obsažen alias zařízení. Zařízení IO device odpovídá kladně paketem Ident OK, pokud je jeho alias shodný.

Poté controller nastavuje síťové rozhraní zařízení IO device, pokud již není nastaveno.

Po nastavení sítě controller zasílá paket Connect request (požadavek připojení viz [1]). V tomto paketu je obsažena kompletní konfigurace IO device, kterou očekává controller. Pokud zařízení splňuje vše, odpoví pozitivním paketem Connect response. Pokud se liší konfigurace modulů/submodulů, je v paketu Connect response obsažen blok ModuleDif-

fBlock, který obsahuje informaci o rozdílu konfigurace controller device.

Poté controller posílá parametrizační data (tzv. Write request viz [1]) a to každému submodulu, pro který jsou definované v GSDML.

Po dokončení konfigurace posílá controller paket Control request Parameter End, kterým signalizuje device ukončení konfigurace. Po odpovědi posílá device paket Application ready, kterým je ukončeno navázání spojení. Dále již běží cyklická komunikace a všechny ostatní služby PROFINETu.

2.1.2 Cyklická komunikace

Cyklická data jsou nepotvrzovaná odesílaná jak z controlleru tak z device v pevném intervalu, který lze nastavit při konfiguraci sítě v controlleru, typickou hodnotou je 1 ms.

Struktura paketu je popsána např. v [1]. Pro každý vstup/výstup je vždy v paketu přítomný blok dat a dále status. Blok dat má velikost proměnnou, ovšem nejmenší jednotkou je jeden bajt. I jednobitové vstupy/výstupy v paketu zaberou celý bajt.

Cyklická data jsou pro čip netX vždy přímo kopírována do DPM do oblasti vstupně výstupních dat a to tak, jak jdou za sebou. Proto i v DPM a ovladači nutné zachovat nejmenší velikost bloku dat jeden bajt, nelze například dva jednobitové vstupy dát do jednoho bajtu. Musí se použít dva bajty.

2.1.3 Acyklická komunikace

Mezi acyklická data se řadí zejména tzv. alarmy. Alarmy slouží k signalizaci chyb nebo poruch, jejich vyčerpávající popis je uveden v [1].

Jsou definovány dva základní alarmy, které jsou dále rozděleny, jedná se o alarmy Hi a Low (vysoká a nízká priorita). Jednotlivé alarmy pro netX jsou popsány v kapitole funkcí poskytovaných nadřazené aplikaci, viz 5.5.2.3.

Alarmy jsou vždy potvrzované a to jak v rámci PROFINETu tak v rámci ovladače.

2.2 GSDML

V rámci vytvoření zařízení PROFINET IO device je nutné vytvořit i jeho popis pro controller. Toto se provádí pomocí tzv. GSDML souborů. Zde bude popsána struktura GSDML souboru a jeho vytvoření pro naše zařízení.

2.2.1 Struktura GSDML

Soubor GSDML je založen na struktuře `xml`. Tato je použita kvůli její snadné zpracovatelnosti programem a také protože již existují hotové nástroje pro práci s xml soubory.

Podrobný popis struktury je uveden v [14].

2.2.2 GSDML pro netX

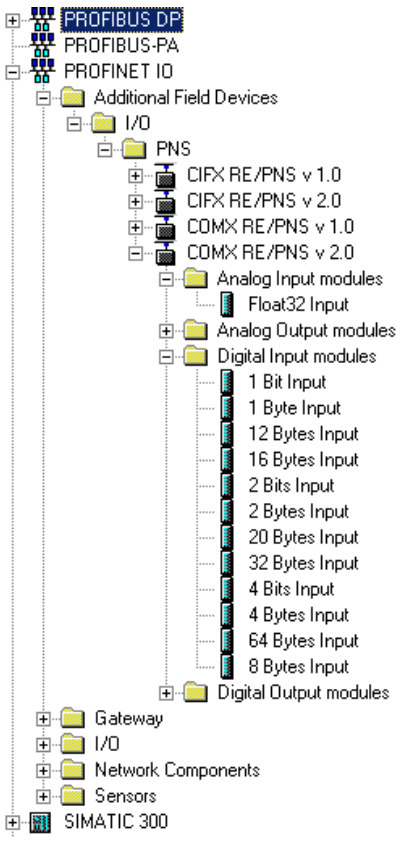
Pro psaní GSDML souborů není třeba žádný speciální software, lze je psát v běžném textovém editoru (jako Vim).

Pro kartu cifX i modul comX firma Hilscher poskytuje soubory GSDML. Tyto však nelze výrazněji použít, protože pro specifikaci GSDML verze 2.0 a vyšší již tuto nesplňuje.

Při tvorbě jsem vyšel z těchto souborů a vytvořil nové GSDML pro oba moduly, tak aby splňovaly GSDML specifikaci. Zejména část Device Access point bylo nutné přepsat.

Pro finální zařízení jsou připraveny dvě verze v každém GSDML (realizováno jako dva různé Device Access point). To je třeba z důvodu použití různých zařízení PROFINET IO controller. Například controller S7-300 neumí pracovat se zařízeními popsanými podle specifikace verze 2 a vyšší. Proto je vždy k dispozici Device Access Point verze 1 pro zařízení starší a verze 2 pro novější. Vyšší verze je nutná při testování zařízení pomocí programu PROFINET IO tester (viz 7.2).

Při použití v programu Step 7 vypadá nabídka podle obrázku 2.4.



Obrázek 2.4: Nabídka v programu Step7

Kapitola 3

Čip netX

V této kapitole bude popsán čip netX. Nejprve jeho struktura a základní vlastnosti, poté bude popsáno rozhraní použité pro komunikaci mezi netXem a hostem.

3.1 Základní informace o čipu

Čip netX je specializovaný komunikační čip vyráběný firmou Hilscher [4]. Všechny zde uvedené údaje jsou platné pro čip netX500. NetX 500 je nejvýkonnější čip, existují ještě tři nižší verze (netX100, netX50 a netX5), které mají omezenou funkčnost. Všechny moduly použité v rámci této práce jsou osazeny čipem netX500, proto bude popsán pouze tento.

Jádrem čipu je 32 bitové CPU **ARM 926EJ-S** běžící na 200MHz s podporou MMU umožňující běh OS. Čip má vlastní paměť typu ROM velikosti 32kB pro bootloader, dále RAM o velikosti 144kB pro data.

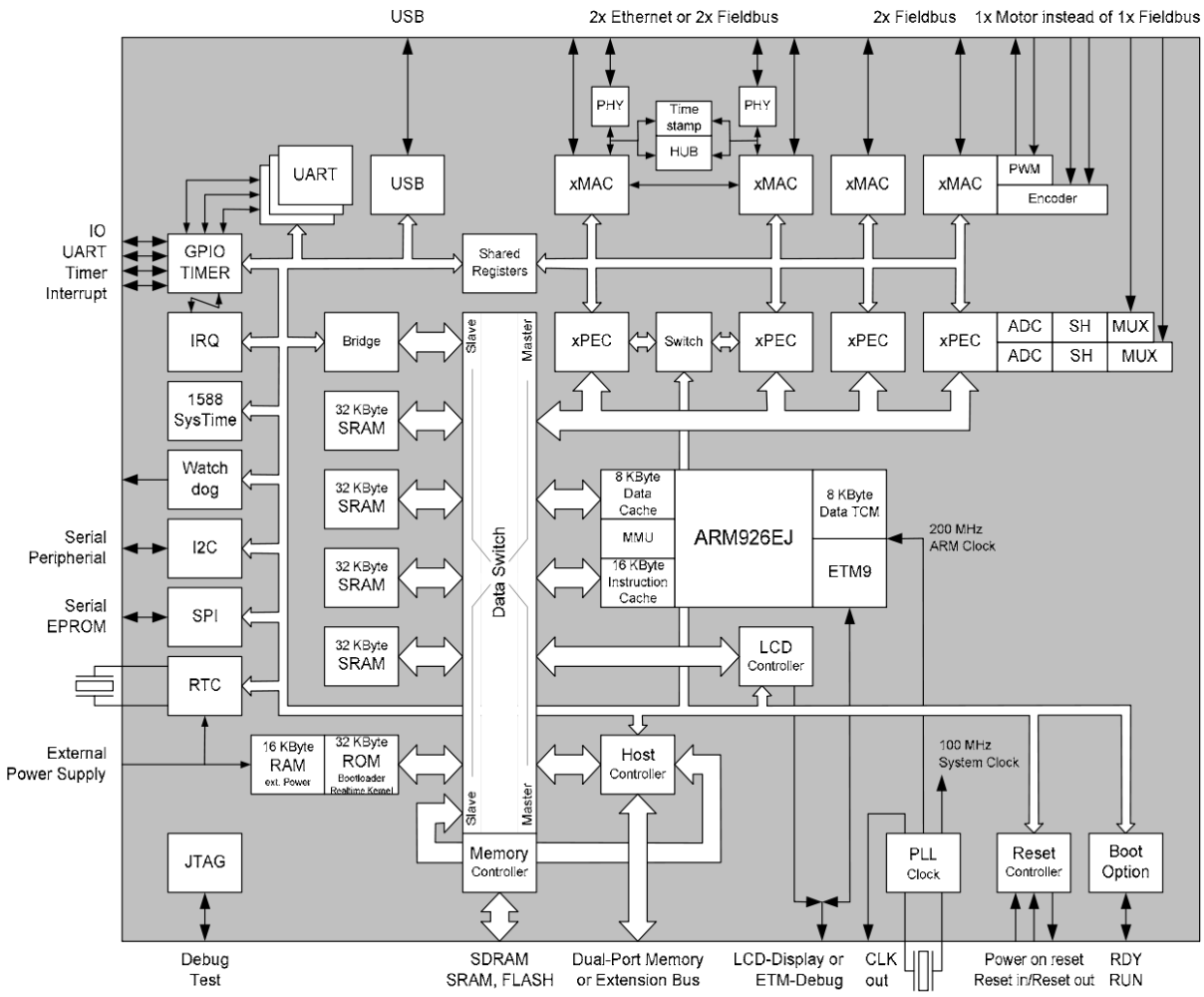
Jako primární rozhraní pro host CPU je připravena dvoubránová paměť. Je-li použit netX jako autonomní, lze vývody nakonfigurovat jako 16bitovou externí sběrnici. Z dalších rozhraní nabízí čip SPI, I2C, USB, UART a JTAG.

Čip je koncipován jako výkonná komunikační periferie při použití s hostem anebo jednodušší autonomní zařízení na fieldbusu.

V rámci této práce je použit jako výkoná komunikační periferie, jejímž cílem je ušetřit hostu všechnu práci se zpracováním stacku fieldbusu a dále umožnit maximální flexibilitu.

3.2 Struktura čipu

Vnitřní struktura čipu je na obrázku 3.1.



Obrázek 3.1: Vnitřní struktura čipu netX (zdroj:[11])

Hlavním prvkem čipu je přepínač kanálů (Data switch), kterým prochází veškerá komunikace. Tento přepínač je navržen tak, aby řadič mohl paralelně přenášet data. V přepínači je vytvořeno pět nezávislých datových sběrnic, tato konfigurace je vytvořena pro maximální propustnost dat bez nutnosti zvyšovat takt sběrnice.

Pro dosažení maximální propustnosti dat je rozdělena vnitřní paměť na 4 bloky po 32kB. Díky tomu může například řadič zapisovat do jedné paměti, host CPU do druhé a xPEC může číst zbylé dvě.

Více informací lze nalézt v [11].

3.3 Software pro čip

Software pro čip není součástí této práce, proto nebude podrobně rozebrán. Program pro netX může být předkompilovaný firmware již obsahující požadovaný stack (od Hilscheru) nebo vlastní.

Při použití předkompilovaného firmware nemáme již žádnou možnost, jak jej ovlivnit. Tuto variantu jsme použili v rámci této práce, protože umožňuje rychlé zprovoznění komunikace a je nejlacinější. Firmware obsahuje OS rcX a stack. Je napsaný pro netX v roli komunikační periferie a využívá DPM pro komunikaci s hostem.

3.4 Dvoubránová paměť (DPM)

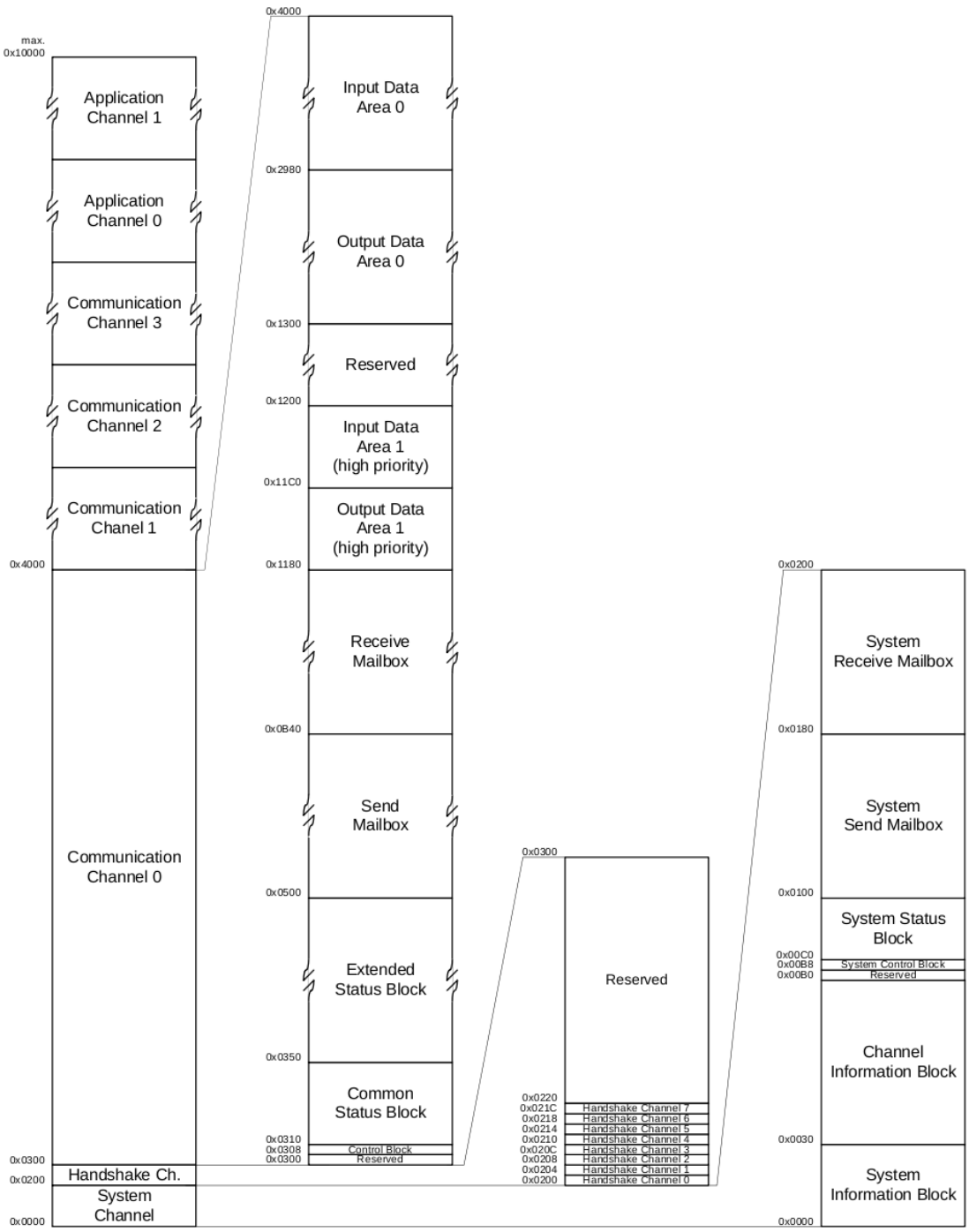
V této kapitole bude popsáno rozhraní poskytované čipem netX. Dvoubránová paměť se z pohledu host CPU jeví jako souvislá paměťová oblast s velikostí 64kB.

Nejprve bude nastíněno rozdělení DPM na jednotlivé funkce a způsoby výměny dat mezi netXem a hostem přes DPM. Poté bude popsána struktura DPM.

3.4.1 Struktura DPM

V této sekci bude popsána struktura dvoubránové paměti. Strukturu lze změnit firmwarem nahraným v netXu, zde uvedený popis platí pro výchozí strukturu DPM, kterou používá předkompilovaný firmware od firmy Hilscher.

Výchozí strukturu DPM lze rozdělit na několik částí. První je tzv. systémový kanál (system channel), což je prostor použitý pro komunikaci mezi aplikací v hostu a čipem netX. Dále tzv. handshake channel, což je prostor použitý pro synchronizaci netXu a hosta. Dále několik tzv. komunikačních kanálů (communication channel), tyto kanály slouží pro komunikaci mezi hostem a použitým firmwarem (obsahujícím protokol fieldbusu). Dále tzv. aplikační kanály, které v této realizaci ovladače nejsou použity. I podle dokumentace od Hilscheru jsou tyto kanály spíše připraveny pro budoucí využití. Na horním konci DPM je umístěn tzv. globální blok registrů (netX Host Register Block). Základní struktura oblastí DPM je na obrázku 3.2.



Obrázek 3.2: Výchozí struktura DPM (zdroj:[10])

Všechny tyto paměťové oblasti jsou podrobně popsány níže.

3.4.1.1 Systémový kanál

Systémový kanál slouží ke komunikaci mezi hostem a operačním systémem v netXu. V čipu běží operační systém v závislosti na použitém firmware. Pro předkompilovaný firmware je tímto OS tav. rcX, což je operační systém přímo od Hilscheru.

Skrze tento kanál probíhá kompletní inicializace a dále veškeré operace, které se týkají celého čipu. Tento kanál je dále rozdělen (viz obrázek 3.2) na následující oblasti.

- Blok systémových informací (System information block)
  V tomto bloku jsou obsaženy základní informace o modulu, jako sériové číslo, zda běží firmware apod. Tato oblast je vyčtena při inicializaci systémového kanálu.

- Blok informacích o kanálech (Channel information block)
  Tato oblast obsahuje popis kanálů implementovaných ve firmwaru. Tento blok je vyčten při inicializaci komunikačních kanálů (viz 5.3.1.2) a podle jeho obsahu jsou vyplněny struktury reprezentující jednotlivé kanály (viz 5.3.1.1). Obsažená data jsou zejména offsety na kterých jsou umístěny jednotlivé části kanálů.

- Systémový kontrolní blok (System control block)
  Tato oblast slouží pro předávání příkazů (na způsob systémového handshake), kam zapisuje host a odkud čte netX. Tímto způsobem jsou předávány například příkazy pro změnu stavu netXu.

- Blok systémového statutu (System status block)
  Tato oblast slouží pro indikaci aktuálního stavu netXu hostu. Obsahuje registr Komunikačního stavu (viz 3.4.2.5), dále několik dalších registrů jako například poslední chybový kód, watchdog a podobně.

- Systémová odesílací schránka (System send mailbox)
  Oblast sloužící k nakopírování dat, které chce host poslat netXu. Tímto způsobem je nahráván například firmware.

- Systémová přijímací schránka (System receive mailbox)
  Funkce shodná s odesílací schránkou, pouze v opačném směru.

Systémový kanál je aktivní zejména při inicializaci a deinicializaci, při běhu komunikace po fieldbusu se nepoužívá.

3.4.1.2 Komunikační kanál

Komunikační kanál slouží ke komunikaci mezi hostem a stackem použitého fieldbusu. Každý komunikační kanál reprezentuje jeden fyzický port na modulu. Komunikační kanál je strukturou podobný systémovému kanálu, oba jsou v rámci ovladače reprezentovány stejnou strukturou (viz 5.3.1.1).

Skrze tento kanál probíhá veškerá komunikace host-fieldbus. Kanál je rozdělen na následující oblasti. Oblast pro vstupní a výstupní data s vysokou prioritou není dosud implementována, proto nebude uvedena.

- Kontrolní blok (Control block)
  Tato oblast slouží pro předávání příkazů (na způsob handshake), kam zapisuje host a odkud čte stack. Tímto způsobem jsou předávány například příkazy pro změnu stavu netXu. V této oblasti je obsažen registr Změn stavu aplikace (viz 3.4.2.6).

- Blok statutu kanálu (Common status block)
  Tato oblast slouží pro indikaci aktuálního stavu stacku hostu. Obsahuje registr Komunikačního stavu (viz 3.4.2.5), dále několik dalších registrů jako například poslední chybový kód, watchdog a podobně.

- Rozšířený statut (Extended status block)
  Oblast obsahující statut stacku, je-li pro zvolený protokol potřeba. Není-li použit, zůstává prázdný.

- Odesílací schránka (Send mailbox)
  Oblast sloužící k nakopírování dat, které chce host poslat stacku.

- Přijímací schránka (Receive mailbox)
  Funkce shodná s odesílací schránkou, pouze v opačném směru.

- Oblast výstupních dat (Output data area)
  Výstupní data jsou ta odesílána netXem na fieldbus ne opačně. Data v této oblasti jsou cyklická data bez režie protokolu (statutu apod.).

- Oblast vstupních dat (Input data area)
  Výstupní data jsou ta přijímána netXem z fieldbusu ne opačně. Data v této oblasti jsou cyklická data bez režie protokolu (statutu apod.).

Tento kanál je zcela neaktivní až do zahájení komunikace po fieldbusu, poté jsou využívány pouze komunikační kanály.

3.4.1.3 Kanál handshake

Tento kanál je zcela výjimečný. Neslouží ke komunikaci, ale pouze k synchronizaci netXu a hosta. Obsahuje registry pro každý kanál. Konkrétně obsahuje registr Systémové příznaky netXu (viz 3.4.2.1), Systémové příznaky hosta (viz 3.4.2.2) dále příznaky netXu a hosta pro každý komunikační kanál (viz 3.4.2.3 a 3.4.2.4).

3.4.1.4 Globální blok registrů

Tento blok není uveden na obrázku 3.2, přesto je vždy přítomen po resetu. Obsahuje registry použité při obsluze přerušení a dále registr použitý při HW resetu. Je umístěn vždy na adrese 0x200 od konce DPM. V [10] není uveden, více informací je pouze v A.3.0.3 v dokumentaci k struktuře `struct NETX_GLOBAL_REG_BLOCKtag`.

3.4.2 Řídící registry

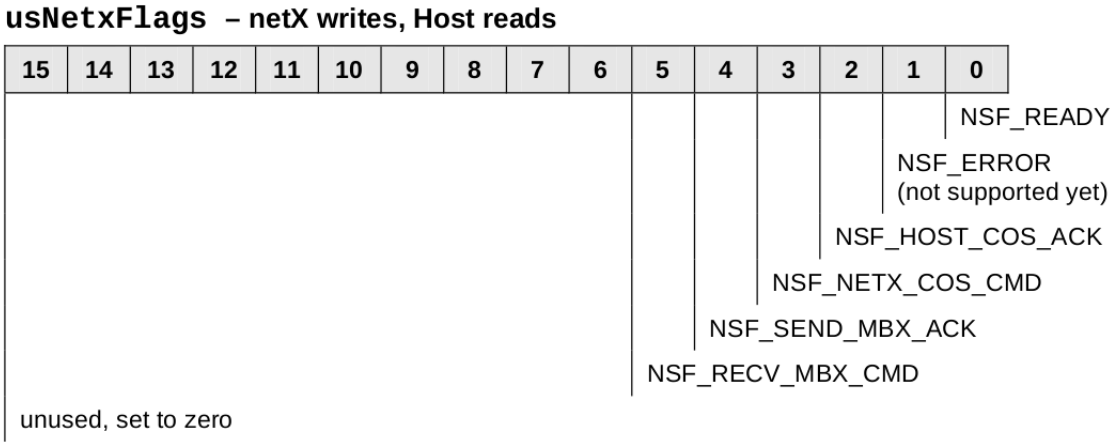
V následujícím textu budou popsány základní registry používané čipem netX. Tyto registry slouží k předávání příkazů mezi hostem a netXem. Všechny níže popsané jsou namapované v DPM.

Pro každý z těchto registrů bude uveden offset v DPM a reprezentace v driveru. Pro popis reprezentace uvažujeme, že `ptDevinstance` je pointer na strukturu DEVICE-INSTANCE příslušející danému zařízení (viz 5.3.1.1). Detailní popis jednotlivých bitů je uveden v [10] a zde nebude opakován.

3.4.2.1 Systémové příznaky netXu

Tento registr se používá k signalizaci stavu netXu hostitelskému počítači. Všechna makra a dokumentaci k tomuto registru lze rozpoznat podle předpony `NSF` (Netx System Flags).

Registr je úzce spjat s registrem systémových příznaků hosta (3.4.2.2), který má shodnou funkci, jen v opačném směru.



Obrázek 3.3: Registr systémových příznaků netXu (zdroj:[10])

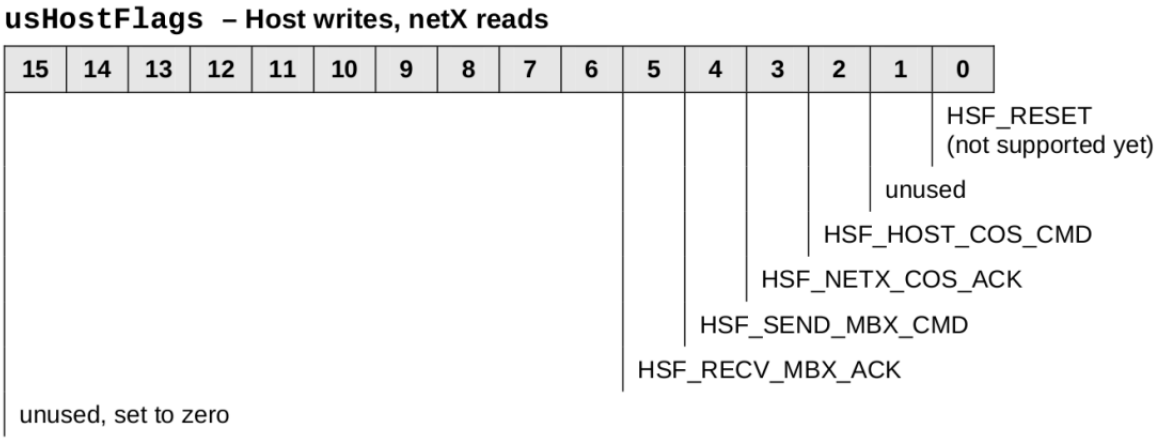
Význam jednotlivých bitů je v [10] na straně 41.

adresa	8 bitový mód: 0x202
	16 bitový mód: 0x200
reprezentace	`DeviceInstance.tSystemDevice.usNetxFlags`

3.4.2.2 Systémové příznaky hosta

Tento registr se používá k signalizaci stavu hostitelského počítače (resp. aplikace) netXu. Všechna makra a dokumentaci k tomuto registru lze rozpoznat podle předpony `HSF` (Host System Flags).

Registr je úzce spjat s předchozím registrem systémových příznaků netXu (3.4.2.1).



Obrázek 3.4: Registr systémových příznaků hosta (zdroj:[10])

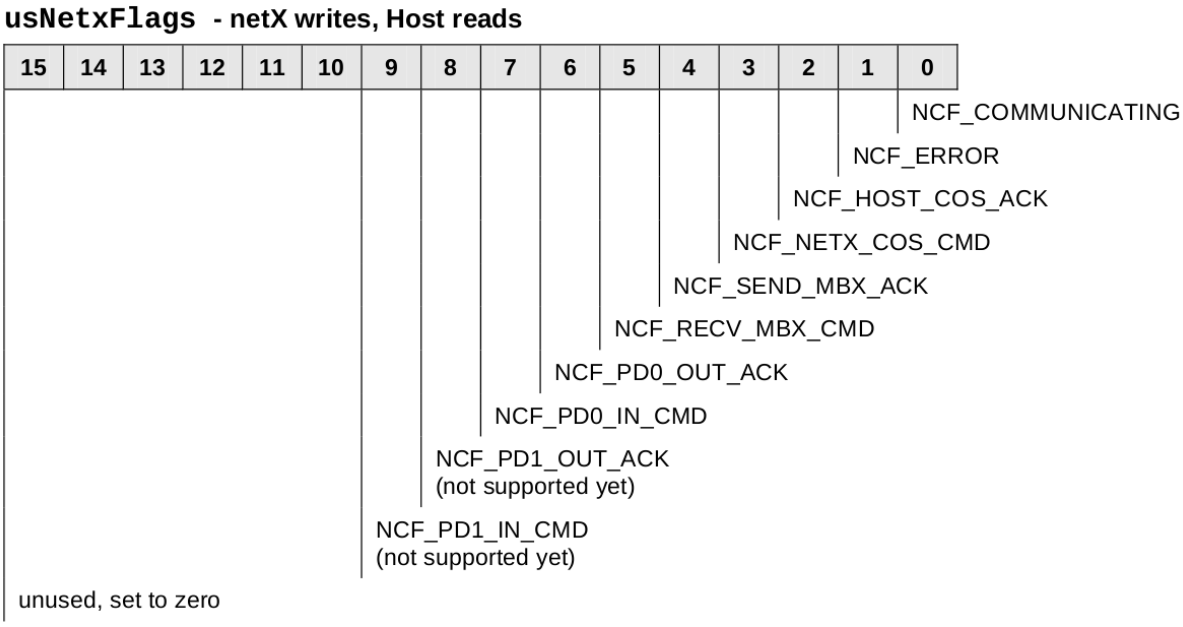
Význam jednotlivých bitů je v [10] na straně 42.

adresa	8 bitový mód: 0x203
	16 bitový mód: 0x202
reprezentace	`DeviceInstance.tSystemDevice.usHostFlags`

3.4.2.3 Komunikační příznaky netXu

Tento registr se používá k signalizaci stavu komunikačního kanálu hostu. Všechna makra a dokumentaci k tomuto registru lze rozpoznat podle předpony `NCF` (Netx Communication Flags).

Registr je úzce spjat s registrem komunikačních příznaků hosta (3.4.2.4).



Obrázek 3.5: Registr komunikačních příznaků stacku (zdroj:[10])

Význam jednotlivých bitů je v [10] na straně 48.

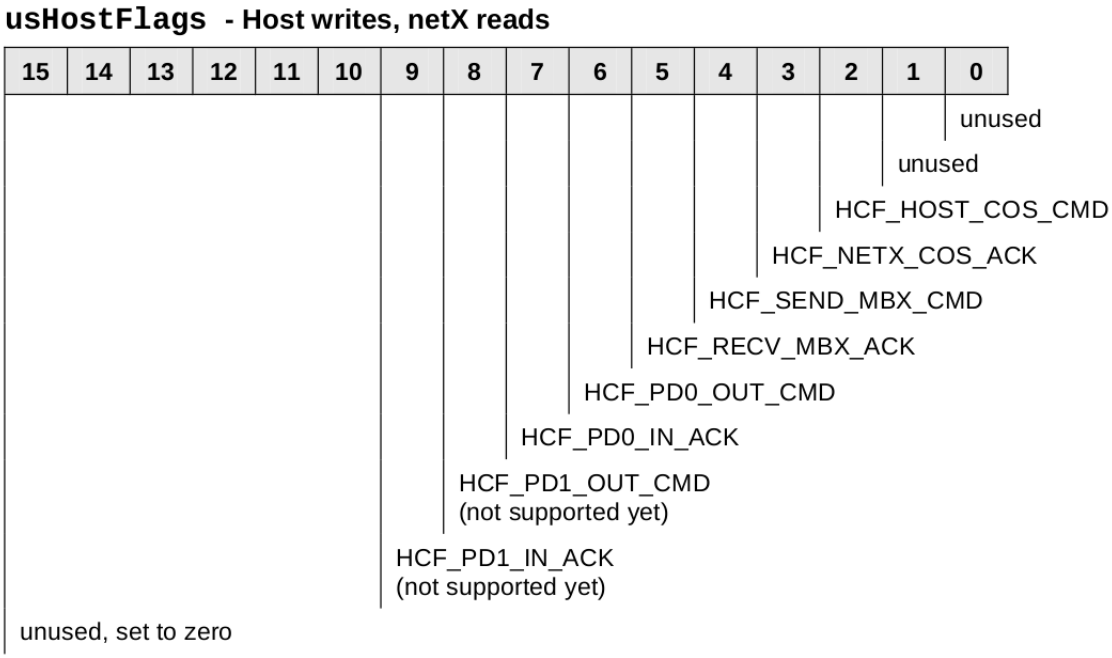
adresa	8 bitový mód:	0x20A	kanál 0
		0x20E	kanál 1
		0x212	kanál 2
		0x216	kanál 3
	16 bitový mód:	0x208	kanál 0
		0x20C	kanál 1
		0x210	kanál 2
		0x214	kanál 3
reprezentace	`DeviceInstance.pptCommChannels[i]->usNetxFlags`		

Kde `i` v reprezentaci je číslo kanálu.

3.4.2.4 Komunikační příznaky hosta

Tento registr se používá k signalizaci stavu hosta komunikačního kanálu . Všechna makra a dokumentaci k tomuto registru lze rozpoznat podle předpony `HCF` (Host Communication Flags).

Registr je úzce spjat s registrem komunikačních příznaků hosta (3.4.2.3).



Obrázek 3.6: Registr komunikačních příznaků hosta (zdroj:[10])

Význam jednotlivých bitů je v [10] na straně 50.

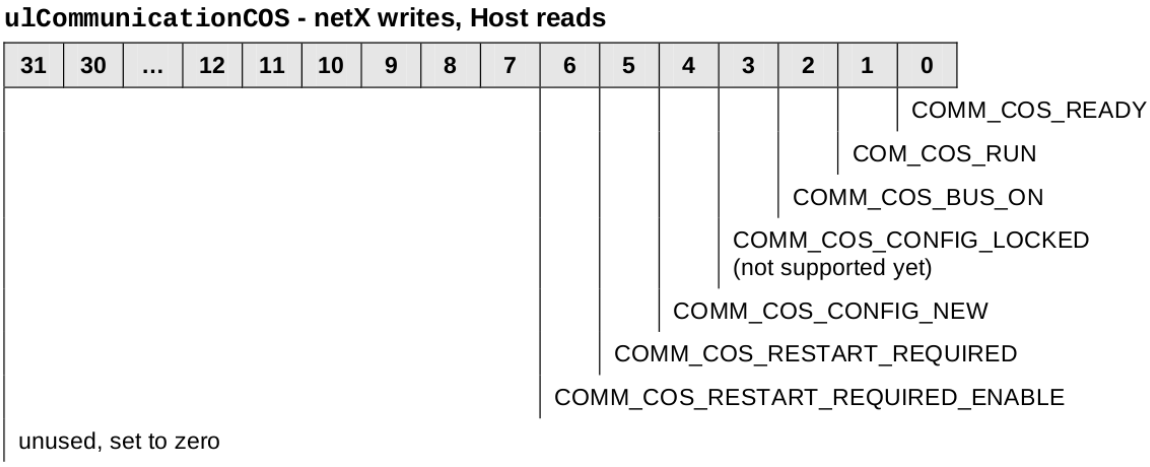
adresa	8 bitový mód:	0x20B	kanál 0
		0x20F	kanál 1
		0x213	kanál 2
		0x217	kanál 3
	16 bitový mód:	0x20A	kanál 0
		0x20E	kanál 1
		0x212	kanál 2
		0x216	kanál 3

reprezentace `DeviceInstance.pptCommChannels[i]->usHostFlags`

Kde `i` v reprezentaci je číslo kanálu.

3.4.2.5 Příznaky změn stavu komunikace

Tento registr se používá k signalizaci změn stavu komunikace, do registru zapisuje stack a čte ho host. Všechna makra a dokumentaci k tomuto registru lze rozpoznat podle předpony `COMM_COS` (COMMunication Change Of State).



Obrázek 3.7: Registr příznaků změn stavu komunikace (zdroj:[10])

Význam jednotlivých bitů je v [10] na straně 56.

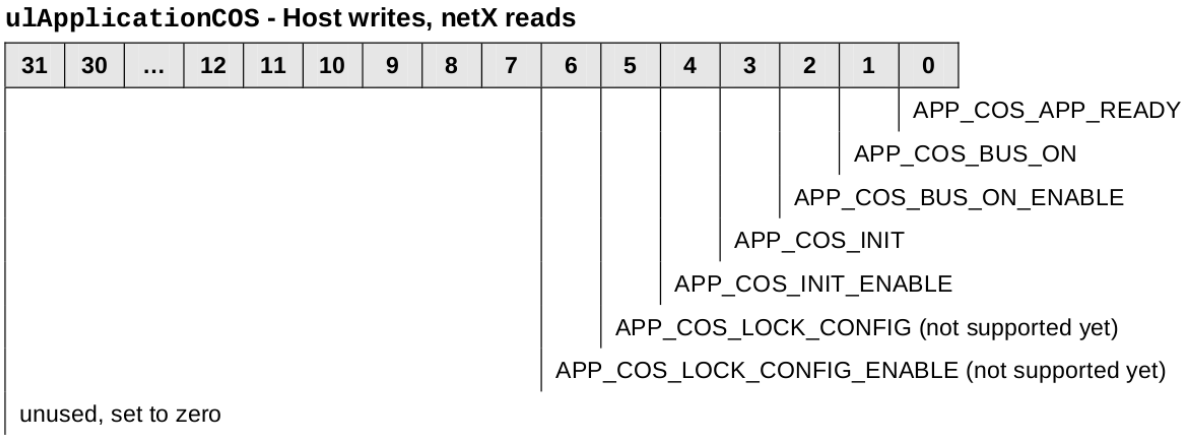
adresa adresa kanálu + 0x10 (0x310 pro kanál 0)

reprezentace

`DeviceInstance.pptCommChannels[i]->ptControlBlock->ulApplicationCOS`

3.4.2.6 Příznaky změn stavu hosta

Tento registr se používá pro poslání příkazů stacku. Všechna makra a dokumentaci k tomuto registru lze rozpoznat podle předpony `APP_COS` (APPlication Change Of State).



Obrázek 3.8: Registr příznaků stavu hosta (zdroj:[10])

Význam jednotlivých bitů je v [10] na straně 53.

adresa adresa kanálu + 0x08 (0x308 pro kanál 0)

reprezentace

`DeviceInstance.pptCommChannels[i]->ptControlBlock->ulApplicationCOS`

Vyčítání hodnot všech výše uvedených registrů je velmi výhodné při ladění aplikace, zejména při inicializaci. Pomocí těchto registrů lze velmi rychle vyčíst aktuální stav netXu.

3.4.3 Přístup do DPM

Pro práci s čipem netX se používají tři typy přístupu do DPM. I přesto se u všech stále jedná o zápis/čtení paměti.

3.4.3.1 Zápis do registrů

Nejprostším typem přístupu je zápis do registrů. Při tomto zápisu měníme hodnoty v registrech popsaných v kapitole 3.4.2.

3.4.3.2 Cyklická data

Při přenosu cyklických dat jsou surová data prostě nakopírována do příslušné oblasti vstupně-výstupních dat. Zápis ani čtení těchto dat **není** nijak **potvrzené**, protože se předpokládá častá práce s cyklickými daty (chyba zápisu se opraví při dalším).

3.4.3.3 Acyklická data

Čtení a zápis acyklických dat probíhá pomocí schránek (mailboxů) a je naopak vždy **potvrzované**. Každá zpráva je zapouzdřena do paketu, který určuje o jakou zprávu se jedná. Toto je zajištěno unifikovanou hlavičkou `TLR_PACKET_HEADER`, kterou obsahuje každý paket. Za hlavičkou jsou obsaženy vlastní data paketu.

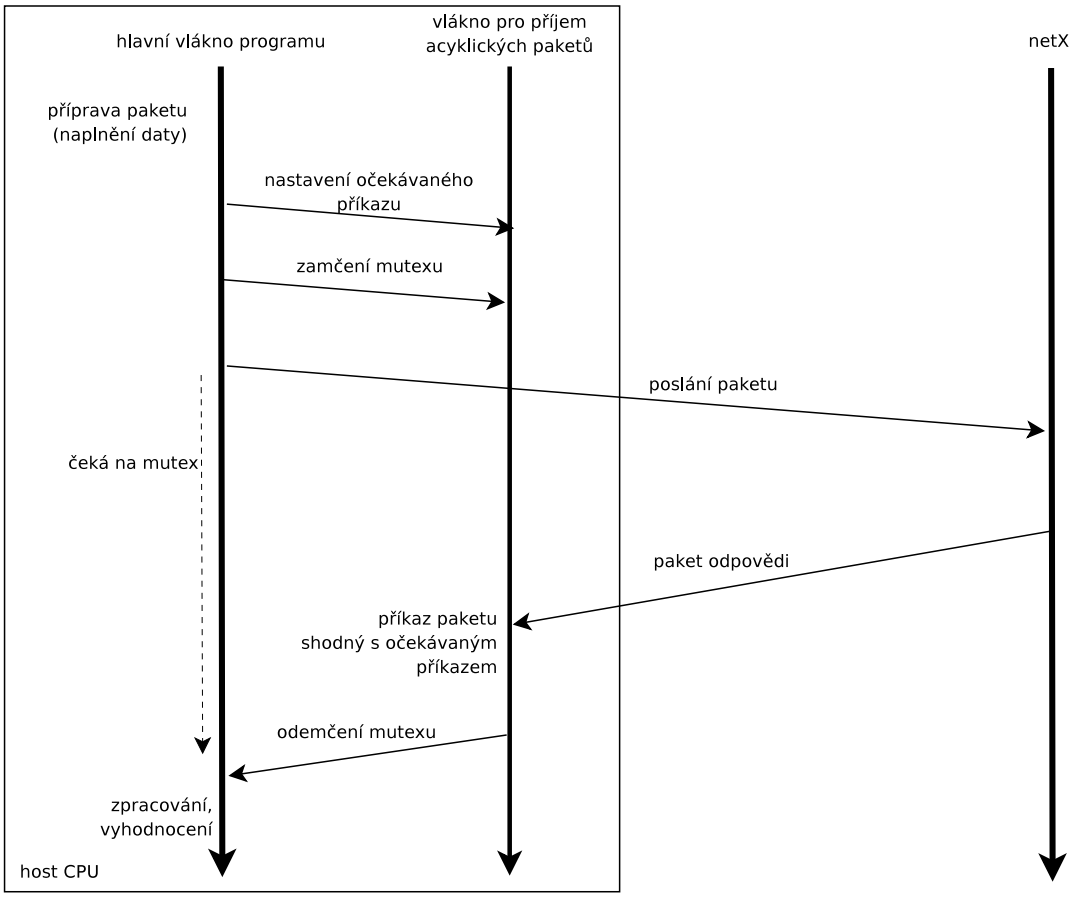
typ	název	popis
unsigned long	ulDest	identifikátor cílového API (typicky 0x20)
unsigned long	ulSrc	identifikátor zdrojového API (typicky 0)
unsigned long	ulDestId	rozšířená identifikace příjemce (typicky 0)
unsigned long	ulSrcId	rozšířená identifikace odesílatele (typicky 0)
unsigned long	ulLen	délka dat paketu (v bajtech)
unsigned long	ulId	pomocný identifikátor paketu (typicky 0)
unsigned long	ulSta	status odpovědi (nese chybový kód ukončení operace)
unsigned long	ulCmd	identifikátor operace, která se má provést (určuje typ paketu)
unsigned long	ulExt	nepoužito, necháno 0
unsigned long	ulRout	nepoužito, necháno 0

Tabulka 3.1: Hlavička acyklických paketů

Acyklická komunikace z pohledu host CPU

Acyklickou komunikaci může zahájit jak host CPU tak netX, je nutné ošetřit obě možnosti. V ovladači byla zvolena varianta s vláknem pro příjem acyklických paketů. Mechanizmus je popsán níže.

- **Komunikaci zahajuje host CPU**
  Při tomto způsobu probíhá činnost podle diagramu na obrázku 3.9.

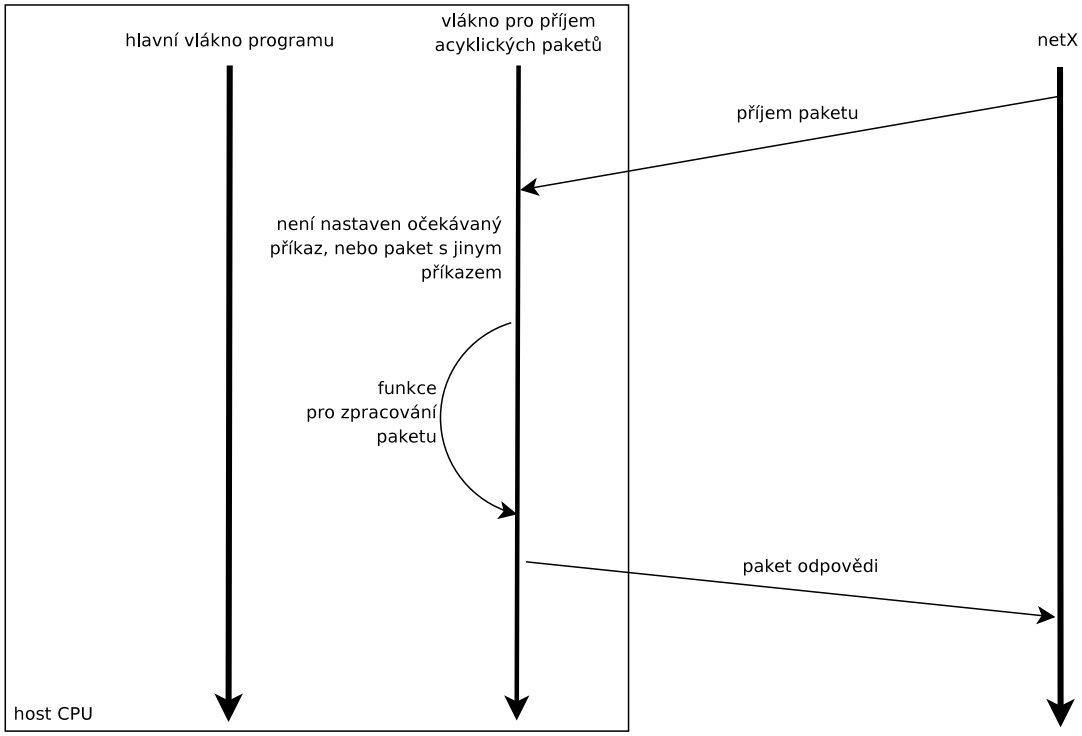


Obrázek 3.9: Průběh acyklické komunikace při zahájení v host CPU

V hlavním vlákně programu připravíme paket. Poté oznámíme vláknu pro příjem, že očekáváme paket odpovědi, toto je realizováno nastavením proměnné očekávaného příkazu (`exp_cmd` viz 5.3.1.1). Je-li tato hodnota nastavena a přijde od netXu paket se shodným příkazem (proměnná `ulCmd` v hlavičce), paket je předán zpět hlavnímu vláknu pro zpracování. Hlavní vlákno mezitím čeká na mutex, který je odemčen při úspěšném přijetí paketu odpovědi.

Pakety tohoto typu lze v manuálu (viz [9]) rozlišit podle přípony pro odesílaný paket `_REQ` a pro paket odpovědi `_RES` (v odpovědi jsou data) nebo `_CNF` (odpověď je pouze potvrzení).

- **Komunikaci zahajuje netX**
  V tomto případě probíhá činnost podle diagramu na obrázku 3.10.



Obrázek 3.10: Průběh acyklické komunikace při zahájení v netXu

Příjem paketu je signalizován vláknu pro příjem. Pokud ve vlákně není nastavena proměnná očekávaného příkazu, nebo je jiná než paketu, je tento paket považován za acyklickou komunikaci zahajovanou čipem netX. Je zavolána funkce pro zpracování paketu (`process_indication(), viz 5.5.2.3`). Tato funkce řeší kompletní zpracování a i odeslání paketu odpovědi.

Pakety tohoto typu lze v manuálu (viz [9]) rozlišit podle přípony pro přijatý paket `_IND` a pro paket odpovědi `_RES`.

3.5 Rozhraní pro PROFINET IO device

Zde bude popsáno rozhraní pro práci s firmware pro PROFINET IO device. Tento firmware je poskytován firmou Hilscher a práce s ním je identická pro všechny varianty ovladače. Kompletní popis viz [9].

Firmware se liší pro kartu cifX a modul comX, nelze je zaměnit. Firmware pro kartu cifX byl přiložen na CD, pro modul comX byl již nahrán firmware verze 2.1.2.0, později byl aktualizován na verzi 2.1.40.0.

3.5.1 Inicializace

Při zprovozňování komunikace po PROFINETu pomocí čipu netX je nejtěžší právě inicializace. Po správné inicializaci je práce z pohledu ovladače jednoduchá.

Zde budou uvedeny dva způsoby inicializace a to manuální inicializace a inicializace pomocí warmstart paketu. Druhá uvedená byla využívána zejména v počátku vývoje ovladače z důvodu jednoduchosti. Nyní je prakticky nepoužita a používá se výhradně manuální inicializace.

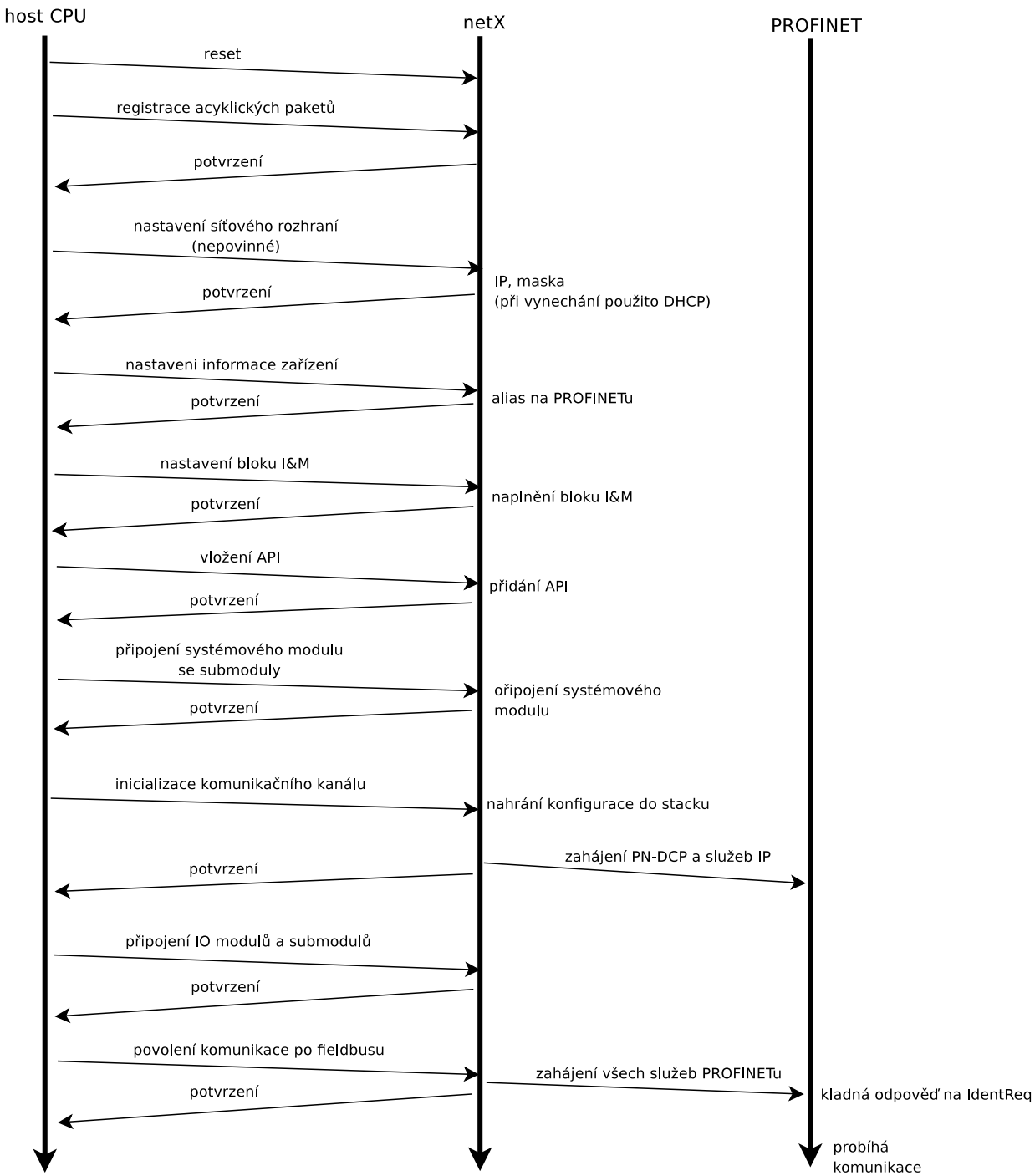
3.5.1.1 Manuální inicializace

V tomto případě celý stack nastavujeme ručně. Toto je základní postup inicializace. Inicializace probíhá podle diagramu na obrázku 3.11.

Průběh inicializace z pohledu netXu probíhá mírně odlišně. Pakety z počátku (nastavení síťového rozhraní až připojení systémového modulu) se neprojevuje okamžitě. Při vyčtení diagnostiky budou všechna nastavení ve výchozím stavu. Tyto pakety jsou uloženy v netXu do speciálního souboru `PCKCFG.PNS`, jeho obsah je načten při inicializaci kanálu a konfigurace je nahrána do stacku.

Soubor `PCKCFG.PNS` nelze zpětně stáhnout z netXu, ani vytvořit v host CPU a poté nahrát.

Všechny operace po inicializaci kanálu se projevují okamžitě po úspěšném dokončení.



Obrázek 3.11: Průběh ruční inicializace

Každý krok inicializace je podrobněji popsán níže. U většiny kroků platí, že jednomu odpovídá jeden acyklický paket.

- **reset**
  Vynulování předchozího nastavení a ukončení veškeré komunikace. Při první inicializaci po spuštění není nutný, netX se automaticky resetuje při spuštění.

- **registrace acyklických paketů**
  Tím zaručíme přeposlání acyklických zpráv z PROFINETu do ovladače.

- **nastavení síťového rozhraní**
  Zde nastavujeme konfiguraci síťového rozhraní pro fieldbus, tj. IP, síťovou masku a výchozí bránu. Nastavení není nutné provést, při jeho přeskočení je použito DHCP.

- **Nastavení informace zařízení**
  Zde přiřazujeme zejména alias na fieldbusu.

- **Nastavení bloku I&M**
  Plníme blok informací I&M (information and maintanence). V netXu v současné verzi je podporováno pouze povinný blok IM0. Více informací o I&M viz [18].

- **Vložení API**
  Je vloženo API, do něj se dále připojují moduly submoduly.

- **Připojení systémového modulu**
  Každé zařízení na PROFINETu musí mít připojen nejméně modul 0 a submodul 1 pro komunikaci. Od verze normy PROFINETu 2.2 jsou povinné další tři submoduly. Jedná se o systémové předdefinované submoduly InterfaceSubmodule a dva PortSubmodule (více informací například viz [14]). Při použití controlleru používajícího starší verzi stacku PROFINETu (např. S7-300) stačí minimální konfigurace (tj. pouze submodul 1).

- **inicializace kanálu**
  Po připojení systémového modulu je provedena inicializace komunikačního kanálu a jsou zahájeny síťové služby TCP/IP a LLDP.

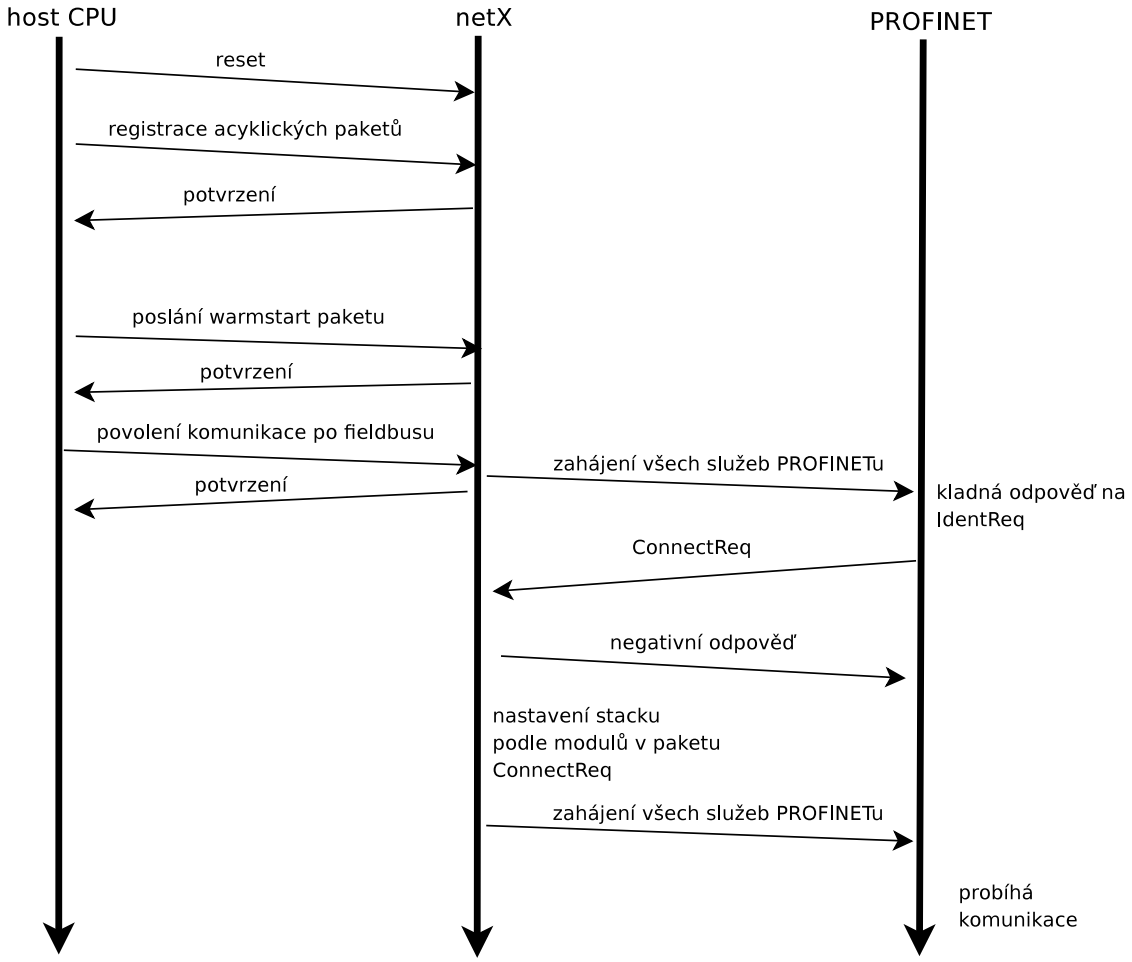
- **Připojení modulů a submodulů**
  V této fázi je třeba postupně připojit všechny požadované moduly a submoduly, aby konfigurace odpovídala hardwarovému uspořádání zařízení.

- **Povolení komunikace na fieldbusu**
  Nakonec jen stačí povolit komunikaci na fieldbusu. Poté již netX čeká na paket Connect request (viz 2.1.1).

3.5.1.2 Inicializace pomocí warmstart paketu

Toto je alternativní varianta konfigurace stacku. Její použití není doporučeno, je uvedena jen pro úplnost.

Její průběh z pohledu host CPU je následující.



Obrázek 3.12: Průběh inicializace pomocí warmstartu

Při použití této inicializace probíhá jinak zahájení komunikace controller-device. Čip netX čeká na Connect request s pouze modulem 0. Po přijetí paketu Connect request vyčte požadovanou konfiguraci, vrátí chybu (negativní Connect response) a poté provede rekonfiguraci stacku, aby odpovídala požadavkům controlleru.

Toto znamená že nemáme **žádnou možnost ovlivnit konfiguraci modulů**. Nelze ani zjistit, které moduly a submoduly jsou připojeny do API. Proto není tato varianta doporučena.

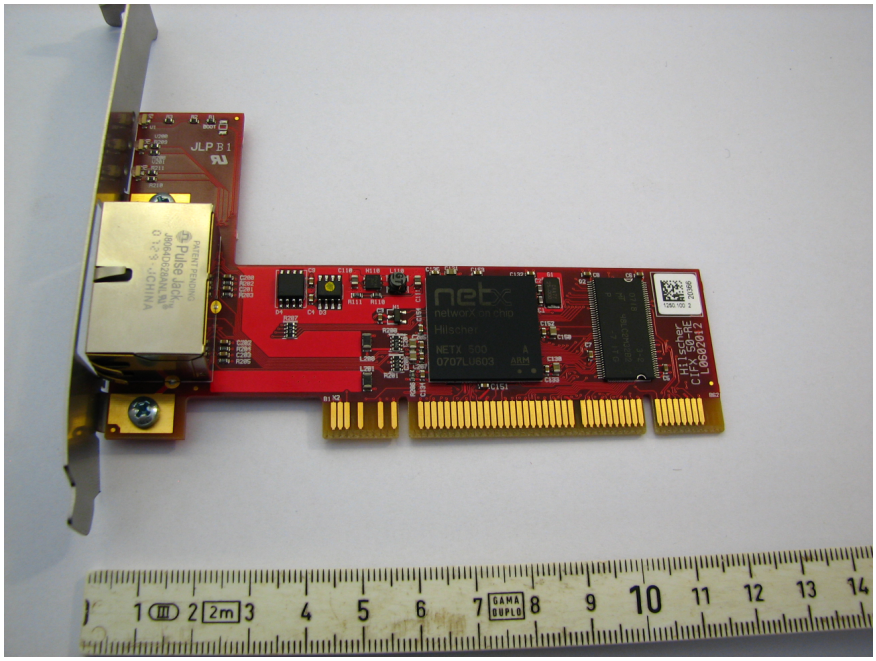
Kapitola 4

Moduly použité pro vývoj

V rámci této práce byly použity dva moduly pro vývoj. Oba dva jsou vyrobeny od Hilscheru a jejich jádrem je čip netX500. V této kapitole bude popsán hardware na kterém probíhal vývoj.

4.1 Karta cifX

Karta cifX je zásuvná karta do rozhraní PCI běžného PC nesoucí čip netX a dva ethernetové konektory pro fieldbus, je na obr. 4.1.



Obrázek 4.1: Karta cifX

Vývoj byl zahájen na této kartě na PC, kde byl vytvořen ovladač pro Linux. Poté byl vývoj přenesen na cílovou platformu, kterou je deska s procesorem MPC5200 s připojeným modulem comX (viz 4.2).

4.1.1 Zprovoznění modulu

Připojení této karty k hostu je triviální, stačí jen připojit do PCI slotu. Jako host je v tomto případě použito běžné PC s operačním systémem Linux (použit Debian 5.0 Lenny).

Pro zprovoznění není třeba žádný speciální software.

Všechny potřebné informace o modulu, jako fyzická adresa, číslo přerušení lze vyčíst z registrů PCI. Napíklad příkazem `lspci -vvv` s výsledkem.

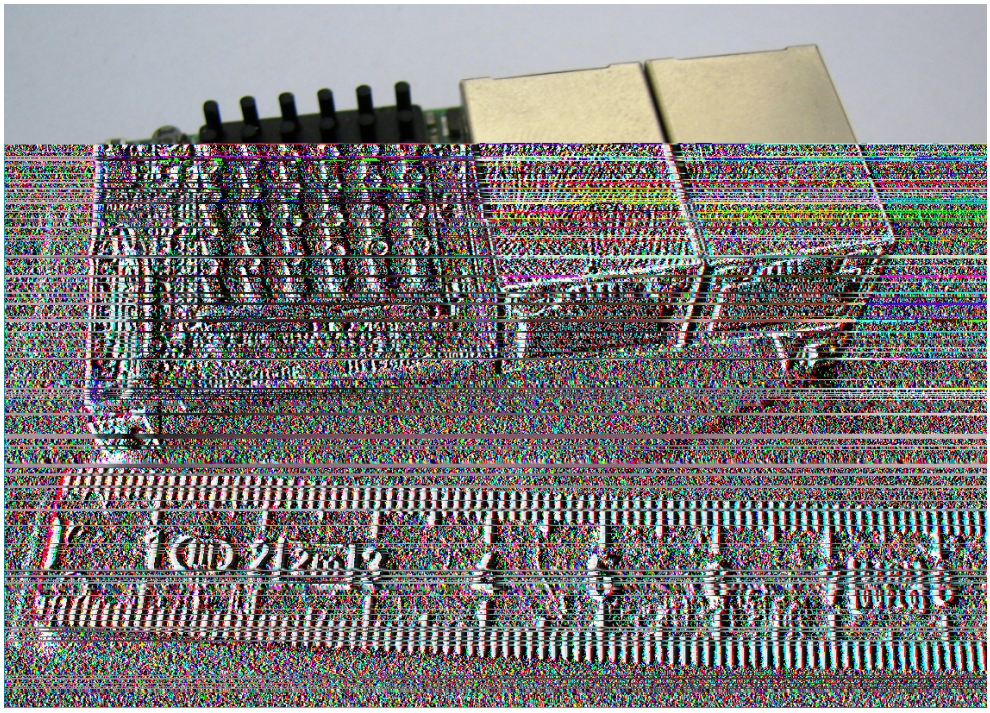
```
00:0d.0 Class ff00: Hilscher GmbH Unknown device 0000
  Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr-
           Stepping- SERR- FastB2B- DisINTx-
  Status: Cap- 66MHz- UDF- FastB2B- ParErr- DEVSEL=medium >TAbort-
           <TAbort- <MAbort- >SERR- <PERR- INTx-
  Latency: 64
  Interrupt: pin A routed to IRQ 5
  Region 0: Memory at dffe0000 (32-bit, non-prefetchable) [size=64K]
  Kernel driver in use: cifX_PCI_driver
```

Poslední řádek (`Kernel driver in use: cifX_PCI_driver`) je uveden až po úspěšném zavedení kernel modulu (viz 5.2.3). Ostatní informace o kartě cifX jsou také vyčítány v kernel modulu.

Ke kartě bylo přiloženo CD s dokumentací, ovladačem pro Windows, firmware a hlavičkovými soubory. Obsah tohoto CD je uveden v příloze a také na přiloženém CD.

4.2 Modul comX

Tento modul je určen pro přímé použití do vestavných zařízení. Je koncipován jako periferie pro host CPU. Má na sobě dva konektory pro Ethernet a konektor pro komunikaci s host CPU.



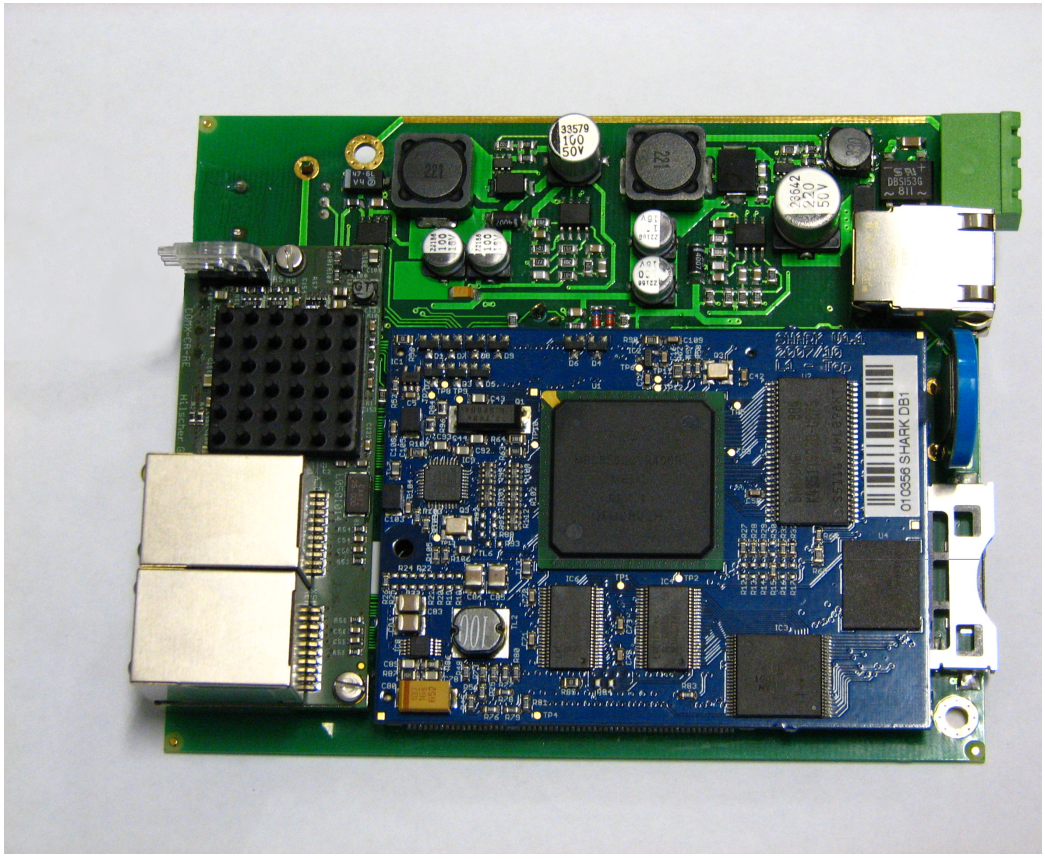
Obrázek 4.2: Modul comX

K tomuto modulu nebylo přiloženo žádné CD ani manuál, což značně ztížilo vývoj. Firmware byl již v modulu nahraný.

V této kapitole bude popsán hardware použitý při práci s modulem comX. Dále zapojení a zprovoznění desky.

4.2.1 Vývojová deska SHARK

Pro potřeby vývoje ovladače byla použita deska SHARK od firmy Mikroklima [6]. Tato deska nese procesorový modul MIDAM s procesorem MPC5200 od firmy Freescale a modul comX. Deska je obrázku 4.3.



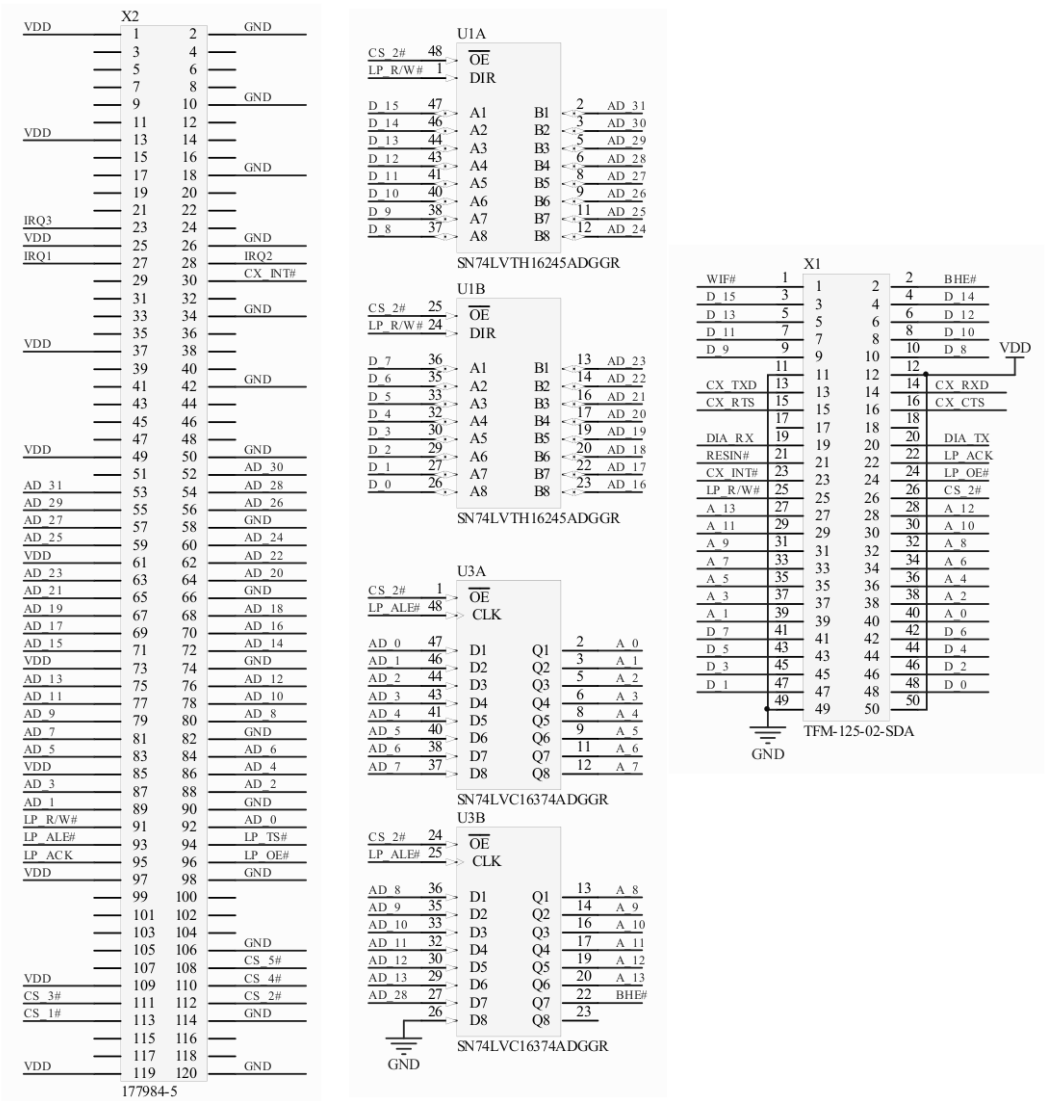
Obrázek 4.3: Deska SHARK

4.2.1.1 Hardwarové zapojení

Zde bude uvedeno hardwarové propojení host CPU s modulem comX, které je použito na desce SHARK. Toto schéma určuje nastavení sběrnice LocalPlus Bus (viz 5.2.4.2).

Propojení modulu a host CPU je realizováno externí datovou sběrnicí LocalPlus Bus. Ta poskytuje až 32 adresně datových vodičů a dále klasické synchronizační vodiče, tj. `R/W`, `CS`, `OE` a `ACK`.

Schéma propojení konektorů na základní desce je na obrázku 4.4.



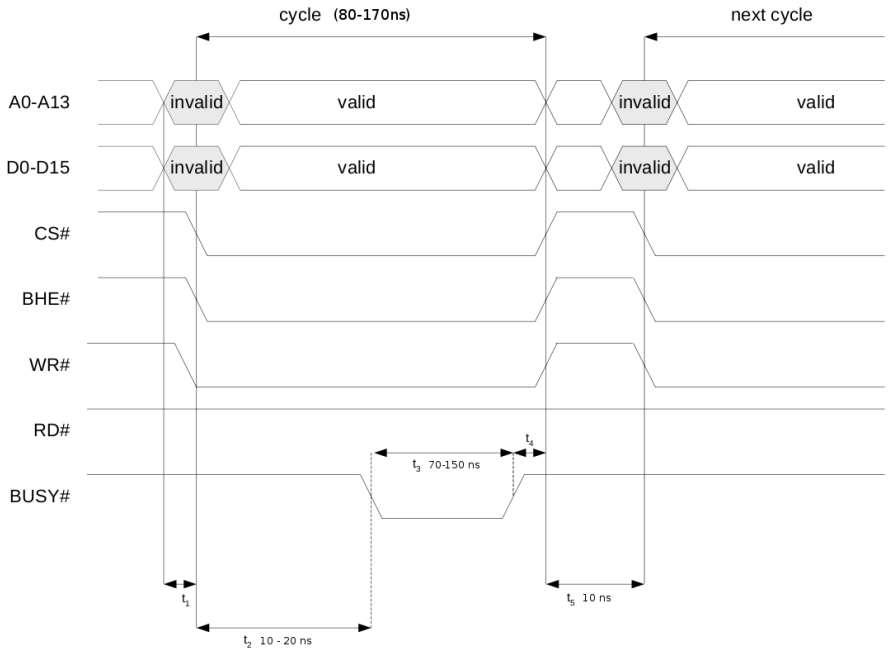
Obrázek 4.4: Propojení na základní desce

Kde X1 je konektor modulu comX a X2 je konektor k desce procesoru MPC5200.

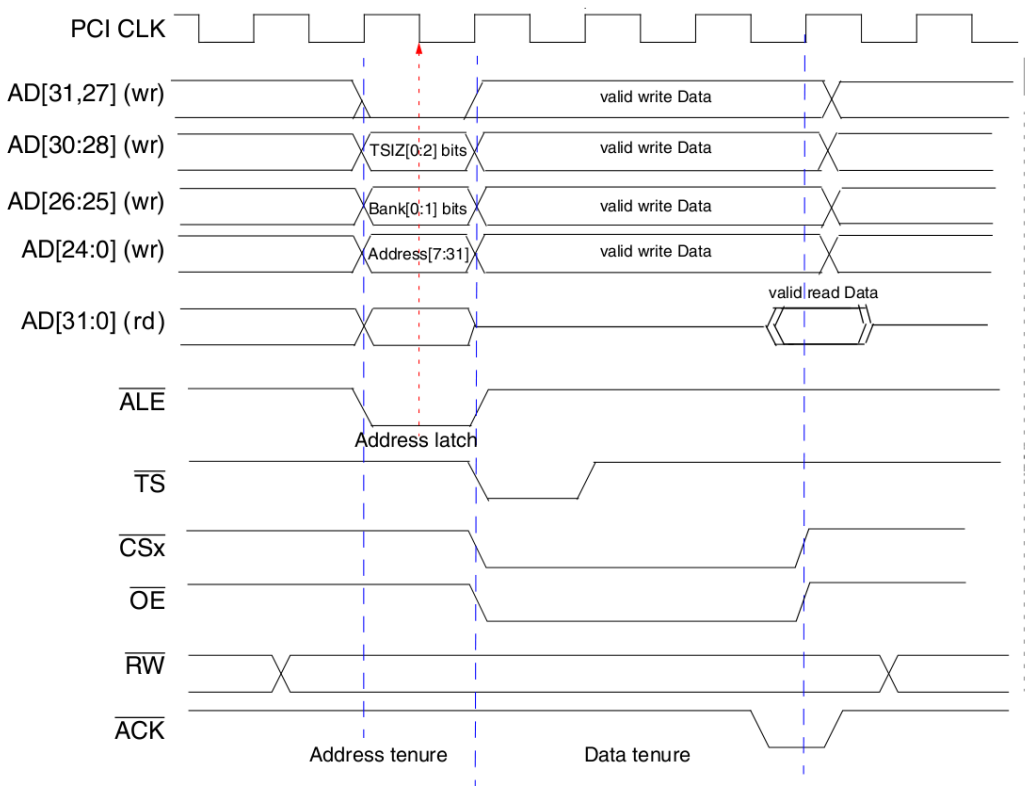
Propojení je realizováno přes vstupně výstupní brány, komunikace probíhá multiplexovaně (funkce viz [8]).

Pro desku SHARK verze 1, je chyba v propojení vodiče ACK na comX. Modul comX používá signál BUSY, zatímco MPC5200 signál ACK, tedy opačnou logiku a nelze tento signál použít pro ověření dokončení cyklu čtení/zápisu. Tento je proto řízen čistě časově a je uvažována jistá rezerva pro zaručení dokončení operace.

Diagramy časování přístupu do paměti pro modul comX a host MPC jsou uvedeny na obrázku 4.5 a 4.6



Obrázek 4.5: Diagramy časování pro comX



Obrázek 4.6: Diagramy časování pro MPC5200

4.2.2 Zprovoznění modulu

Pro tuto variantu je samotné zprovoznění vývojové desky celkem komplikované. Popis je uveden v [19].

Pro zprovoznění desky SHARK je třeba pročíst zejména stránku věnovanou modulu MPC5200, kde jsou uvedeny konkrétní informace o použité desce, včetně rozběhnutí OS. Zprovoznění služeb potřebných v PC je popsáno na stránce o desce Boa5200 HOWTO, jedná se o TFTP, DHCP a NFS.

V rámci této práce byla použita verze Linuxového jádra 2.6.26.5-00006-g7fa3cc3 a PikeOS verze 2.2.

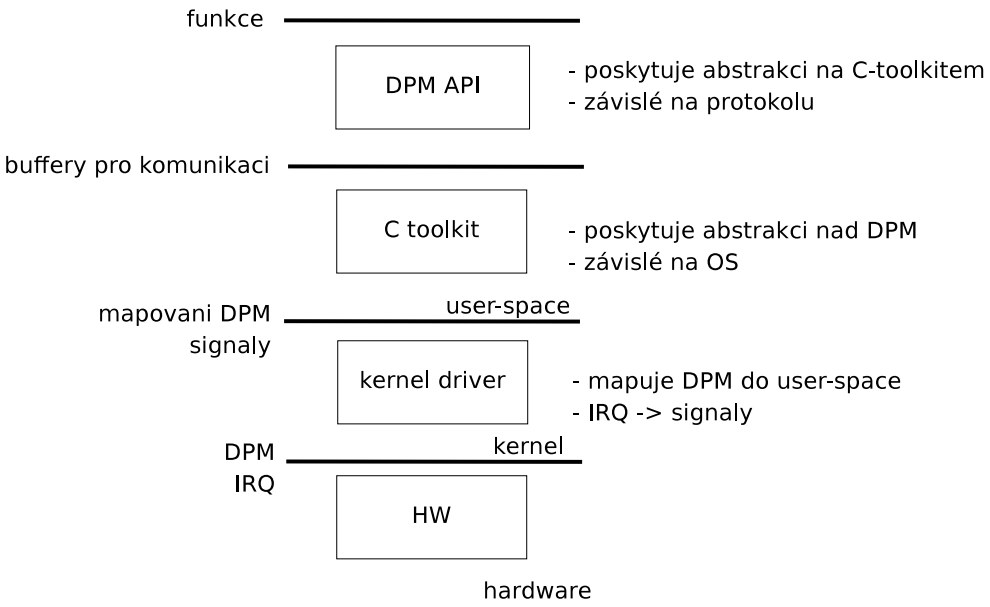
Kapitola 5

Ovladač pro Linux

V této kapitole bude popsán ovladač vyvinutý pro OS Linux. Tento ovladač má dvě verze, první pro PC a druhou pro powerPC MPC5200, zde budou popsány obě. Na tuto kapitolu se také odkazuje kapitola o ovladači pro PikeOS (6) a to v částech mimo kernel, které jsou téměř totožné.

5.1 Struktura ovladače

Strukturu lze nejsnáze vyčíst z obrázku 5.1



Obrázek 5.1: Struktura ovladače

Celý ovladač je rozdělen do tří částí a to kernel modul, C-toolkit a DPM API. Smyslem tohoto rozdělení je maximální flexibilita ovladače, protože každá část má oddělenou funkci.

Z obrázku 5.1 lze odhadnout význam jednotlivých částí.

Kernel modul je zodpovědný za zapouzdření hardwarového propojení hosta s netXem a částem výše musí poskytnout namapovanou DPM. Při změně hardware nebo OS musí být celý přepsán.

C-toolkit je zodpovědný za zapouzdření OS. Od kernel modulu očekává namapovanou DPM jako souvislou paměťovou oblast. Pro API poskytuje sadu funkcí potřebných pro práci s DPM, také zapouzdřuje strukturu DPM, se kterou se vyšší část aplikace nemusí vůbec zabývat.

DPM API je nejvyšší část ovladače. Na této úrovni se řeší část závislá pouze na použitém komunikačním protokolu. Tato část očekává vždy stejné rozhraní od C-toolkitu bez ohledu na použitý hardware nebo operační systém.

Detailní popis jednotlivých částí je v 5.2, 5.3, 5.4 a 5.5.

Kernel modul je samostatný a musí být do jádra zaveden ručně (případně automaticky při startu). C-toolkit společně s DPM API jsou zkompilovány do objektů `*.o`, které jsou připraveny k slinkování s cílovou aplikací.

Pro potřeby aplikace jsou připraveny hlavičkové soubory `DPM_api.h` a `PN_api.h` (viz A.1), které obsahují veškeré funkce potřebné pro práci s netXem.

5.2 Kernel modul

Kernel modul řeší veškeré problémy spojené s hardwarem použité platformy. Pro vyšší části ovladače musí být přístup do DPM zcela transparentní. V tomto dokumentu bude popsán kernel modul pro PCI kartu do PC a pro LocalPlus Bus na MPC5200 od firmy Freescale [5]. Vytvoření kernel modulu tu nebude vysvětleno, pro informace o kernel modulech do Linuxu viz [7].

Modul je napsán jako znakové zařízení, tato volba však není nijak významná. Typické funkce znakového zařízení jako `read()` a `write()` nejsou vůbec použity. Přístup přes mapování paměti je pro práci s DPM netXu naprosto přirozený a také nejrychlejší.

Cílem této kapitoly není podrobně popisovat jednotlivé funkce, pro tuto potřebu je vygenerovaná dokumentace pomocí Doxygenu A.3.0.1 a A.3.0.2 v příloze. Cílem je popsat funkci modulu jako celku.

5.2.1 Část povinná pro modul

Zde bude popsána část povinná pro modul.

5.2.1.1 `init()`

Vstupní bod modulu, tato funkce je zodpovědná za následující operace

- Dynamické přiřazení MAJOR. Při alokaci se vytvoří záznam v `/proc/devices`, odkud získané MAJOR číslo vyčítáme. Toto číslo také musí být použito při vytvoření zařízení v `/dev`. Bez tohoto nelze úspěšně modul použít.
- Alokovat paměť pro datovou strukturu modulu.
- Zinicializovat strukturu `cdev`. Tímto krokem přiřadíme volání modulu funkcím našeho ovladače.

V rámci této funkce může být nutné provést více operací, tyto jsou ale vždy nutné.

5.2.1.2 `exit()`

Tato funkce je poslední volaná pro modul. Má na starost modulu, zejména odstranění všeho co bylo vytvořeno ve funkci `init()`.

- Odmapování DPM
- Uvolnění paměti
- Uvolnění MAJOR čísla

5.2.1.3 `open()`

Tato funkce nemá žádné povinné operace.

5.2.1.4 `release()`

Tato funkce nemá žádné povinné operace.

5.2.1.5 `ioctl()`

Tato funkce je použita pro veškeré operace vymykající se operacím definovaným v `nX_fops` (viz A.3.0.1). Pro základní zprovoznění není nutné implementovat, funkce jsou nutné pro inicializaci a zpracování přerušení. Pro potřeby přerušení jsou použity následující volání.

- `add_pid()`
  Tato funkce slouží k uložení PID aplikace v user-space v paměti modulu. PID je využit při posílání signálu, kde je vyžadován. Jako signál je použit SIGRTMIN+1. Čísla PID jsou ukládána do spojového seznamu a jejich počet není omezen.

- `rm_pid()`
  Opačná funkce k předchozí, tímto voláním se smaže daný PID a aplikace již není informována signálem o příchodu přerušení.

5.2.2 Část vyžadovaná částí ovladače v user-space

5.2.2.1 `mmap()`

Tato funkce zprostředkovává klíčovou funkci kernel modulu přemapování fyzické adresy do adresního prostoru user-space. V rámci této funkce je třeba provést následující kroky.

- Ověřit rozsah adres.
  Kernel modul nesmí umožnit namapovat adresy mimo rozsah DPM. Offset v argumentu funkce udává počáteční fyzickou adresu, ne ovšem absolutní, ale pouze uvnitř prostoru DPM. Při práci s DPM netXu vždy mapujeme celou paměť, proto offset je vždy 0 a velikost shodná s rozsahem DPM. Aplikace v user-space nepotřebuje znát aktuální fyzickou adresu a její vyčítání pouze zvyšuje složitost. Absolutní fyzickou adresu zná pouze kernel modul, ten ji přičítá k offsetu.

- Nastavit příznaky přístupu.
  Přístup do DPM musí být nekešovaný, dále musí být oblast nastavena jako IO paměť.

- Přemapovat DPM.
  Je-li rozsah adres platný a jsou nastaveny příznaky, provede se vlastní přemapování.

5.2.2.2 `isr()`

Obslužná rutina přerušení, je volána vždy, když netX vyvolá přerušení. V rámci této funkce se vykonávají následující operace.

- Zkopírovat handshake kanálu, zde je obsažen zdroj přerušení.
- Zpětně potvrdit přerušení netXu a tím zastavit jeho opakování. Bez tohoto kroku netX stále opakuje přerušení, což zcela zablokuje činnost hosta.
- Poslat signál aplikaci v user-space. Signál je poslán všem úlohám, jejichž PID je v modulu uložen.

5.2.3 Část povinná pro PCI

Při použití modulu cifX (karta PCI) je nutné v kernel modulu implementovat obslužné rutiny pro práci s PCI. Všechny nutné funkce jsou popsány níže. Dále je zde uvedeno rozšíření funkcí již uvedených.

5.2.3.1 datová struktura popisující ovladač PCI

Pro určení který ovladač použít pro HW je nunté deklarovat datovou strukturu přiřazující základní informace.

5.2.3.2 `init()`

V této funkci je nutné přidat příkaz na registraci ovladače pro PCI kartu.

5.2.3.3 `probe()`

Tato funkce se volá ihned po připojení karty do PCI slotu. V našem případě je tato funkce volána okamžitě po registraci ovladače ve funkci `init()`, protože karta je přítomna již při startu PC.

Tato funkce musí provést následující kroky.

- Zaregistrovat si výlučný přístup do rozsahu paměti pro DPM.
  Tímto se zabrání přístupu jiného programu do DPM.
- Namapovat DPM do adresního prostoru jádra.
  Kernel modul při komunikaci nepřistupuje do DPM. Pouze v ISR potřebuje zapsat do synchronizačních registrů netX.

- Vyčíst číslo IRQ a zaregistrovat ISR
  Číslo IRQ se vyčítá z konfiguračního registru PCI. Poté je zaregistrováno ISR, přerušení musí být registrováno jako sdílené.

- Povolit zařízení
  Po tomto volání je již zařízení připraveno k běhu.

5.2.3.4 `remove()`

Tato funkce je doplňková k `probe`, má za úkol zrušit vše, co se provedlo v `probe()`. Konkrétně tyto kroky.

- Uvolnit IRQ, resp. zrušit registraci ISR.
- Odmapovat DPM z adresního prostoru jádra.
- Uvolnit rozsah adres použitý pro DPM.
- Zakázat běh zařízení.

5.2.3.5 `isr()`

Všechna přerušení na PCI jsou sdílená. Jako první musí ISR určit, zda se jedná o přerušení od karty cifX. Ovladač zkontroluje registry netXu, zda vyvolal přerušení. Pokud ne, ISR skončí a nechá přerušení nezpracované. V opačném případě provede operace popsané v 5.2.2.2.

5.2.3.6 `ioctl()`

Při inicializaci modulu cifX je netX vždy zresetován. Při resetu jsou zneplatněny konfigurační registry PCI. Karta cifX proto vyžaduje funkce pro zápis a čtení konfiguračních registrů.

- `read_PCI()`
  Tato funkce přečte celý konfigurační registr a vrátí jej do user-space části ovladače.

- `write_PCI()`
  Tato funkce zapíše data z user-space do konfiguračního registru karty cifX. Tyto data musí obsahovat platný registr, jinak je funkce karty znemožněna. V našem případě se jedná o data předtím vyčtená z registru, tedy pokud nejsou nijak upravována nehrozí žádná chyba.

5.2.4 Část povinná pro LocalPlus Bus

Při použití modulu comX je nutné nastavit komunikace mezi host CPU (MPC5200) a modulem comX. Popis hardwarového propojení je v 4.2.1.1.

5.2.4.1 `init()`

Tuto funkci je oproti popisu v 5.2.1.1 nutné rozšířit o následující operace.

- Namapovat registry procesoru do adresního prostoru jádra.
  Toto je nutné pro nastavení LocalPlus Bus sběrnice. V rámci modulu se pracuje s registry od MBAR (viz [8]) až do MBAR+0x510.

- Namapovat DPM do adresního prostoru jádra.
  Kernel modul přistupuje do DPM pouze kvůli synchronizaci, jinak nikoli.

5.2.4.2 `open()`

V rámci této funkce je třeba nastavit periférie procesoru. Tato nastavení zasahují do registrů jejichž popis lze nalézt v [8]. Kompletní postup co je třeba je uveden níže, všechny kroky jsou nutné pro funkci modulu.

- Zakázat PCI sběrnici.
  PCI sběrnice sdílí některé vývody se sběrnicí LocalPlus Bus a nelze je provozovat paralelně. Registr MBAR+0xB00.

- Zakázat práci se signálem chip select (CS2).
  Při přenastavení CS2 je nutné jej zakázat, jinak může dojít k zápisu do neplatné paměti. Registr MBAR+0x308 (registr CS2).

- Zakázat čtení a zápis v BURST módu pro CS2.
  Modul comX tento přístup nepodporuje. Registr MBAR+0x328.

- Nastavit rozsah adres pro modul comX.
  Hodnoty jsou horních 16 bitů z adres. Registry MBAR+0x14 a MBAR+0x18.

- Povolit CS2 v IPBI registru.
  Registr MBAR+0x54.

- Povolit CS2 v registru pro všechny CS.
  Registr MBAR+0x318.

- Nastavit registr CS2.
  Zde se nastaví komunikace po sběrnici LocalPlus Bus s použitím signálu CS2. Nastavení jednotlivých bitů vychází z hardwarového zapojení. Hodnoty jednotlivých bitů jsou zde uvedeny, podrobné vysvětlení jejich funkce viz [8]. Registr MBAR+308.
  - **WaitP = 0x9** Počet čekacích cyklů ($f_{CLK} = 66\,\mathrm{MHz}$).
  - **WaitX = 0x9** Počet čekacích cyklů, shodné jako WaitP.
  - **MX = 1** Multiplexovaná komunikace.
  - **AA = 0** Nepoužívat signál ACK pro ukončení cyklu. Pro desku verze 1.0 nelze použít, v novější by již měla být přidána podpora pro signál ACK.
  - **CE = 1** Povolení CS2.
  - **AS = 1** Velikost adres, 0x1 odpovídá 16 bitům.
  - **DS = 1** Velikost dat, 0x1 odpovídá 16 bitům.
  - **Bank = 0** Řídí adresní signálu A_{25} a A_{26}, tyto nepoužíváme.
  - **Wtyp = 1** Určuje použití WaitP a WaitX, pro 0x1 se WaitP použije pro čtení a WaitX pro zápis.
  - **WS = 1** Protočení pořadí bajtů (byte order). Nutné jelikož netX je little endian a MPC5200 je big endian a dále velikost dat je větší než bajt.
  - **RS = 1** Shodné s WS.
  - **WO = 0** Při nastavení 0x1 by byl povolen pouze zápis.
  - **RO = 0** Shodné s WO.
- Povolit příjem přerušení z periférií.
  Modul comX je zapojen jako přerušení IRQ0, registr MBAR+0x510.

5.2.4.3 `release()`

V této funkci nastavíme registry upravené ve funkci `open()` na výchozí hodnoty, přinejmenším je nutné zakázat přístup na sběrnici LocalPlus Bus s použitím CS2.

5.3 C-toolkit pro PC

C-toolkit je ve výchozí podobě sada funkcí poskytovaná firmou Hilscher. Lze jej stáhnout na jejich webových stránkách (viz [4]).

Je to pouze sada funkcí, tak jak je poskytován jej nelze nijak použít. Pro snadnější popis jej rozdělíme do tří částí.

5.3.1 Hlavní část

První je hlavní část, která implementuje většinu obslužných funkcí nad DPM. Ty jsou dále použity v DPM API (viz 5.5). Tyto funkce budou popsány níže. Tato část toolkitu je nejrozsáhlejší a není ani celá použita.

Při použití PCI modulu cifX není nutné hlavní část nijak výrazně upravit. Drobné úpravy je nutné provést pro zpracování acyklických volání iniciovaných čipem netX.

Kompletní popis je nad rozsah této práce, pro bližší informace viz A.3.0.3.

5.3.1.1 Datové struktury

Zde budou popsány základní datové struktury použité v hlavní části C-toolkitu. Obě struktury jsou použity často v celém ovladači, proto je jejich popis potřeba pro pochopení funkce ovladače.

Struktura DEVICEINSTANCE

Tato struktura v sobě obsahuje informace o jednom zařízení. Zahrnuje instance všech kanálů (systémový i všechny komunikační). Výčet nejdůležitějších prvků je uveden níže (není kompletní, ten viz A.3.0.3).

typ	název	popis
unsigned long	ulPhysicalAddress	Fyzická adresa začátku DPM (vždy 0)
unsigned char	bIrqNumber	číslo IRQ přiřazené čipu netX
int	fIrqEnabled	jedna, pokud používáme přerušení
int	fPCICard	jedna pokud je modulem PCI karta
void*	pvOSDependent	ukazatel na identifikaci zařízení (filedescriptor)
unsigned char*	pbDPM	adresa začátku DPM v paměťovém prostoru user-space
unsigned lonf	ulDPMSize	velikost DPM
char	szName[]	název zařízení (typicky v `/dev`)
char	szAlias[]	alias netXu na fieldbusu
unsigned long	ulSerialNumber	sériové číslo karty
unsigned long	ulDeviceNumber	výrobní číslo karty
CHANNELINSTANCE	tSystemDevice	struktura pro systémový kanál
unsigned long	ulCommChannelCount	počet komunikačních kanálů
CHANNELINSTANCE**	pptCommChannels	pole struktur pro komunikační kanály

Tabulka 5.1: Výčet prvků struktury DEVICEINSTANCE

Struktura CHANNELINSTANCE

Tato struktura v sobě obsahuje reprezentaci jednoho kanálu. Ten může být buď systémový nebo komunikační. Výčet nejdůležitějších prvků je níže, kompletní je viz A.3.0.3.

Do této struktury je navíc přidáno několik proměnných pro příjem acyklických zpráv od netX (funkce viz 3.4.3.3). Tyto proměnné jsou

- `ac_sync`
  Synchronizační primitivum, realizováno jako mutex (z knihovny `pthread.h`). Při přijetí paketu je signalizováno odemknutím mutexu.

- `pPacket`
  Přijatý paket je zkopírován do tohoto bufferu pro zpracování.

- `ul_cmd_exp`
  Pokud čeká host potvrzení příkazu, nastaví do této proměnné očekávaný příkaz. Pokud neprobíhá žádná acyklická komunikace ze strany hosta, má hodnotu -1. Poté je přijatý paket zpracován jako indikace od netXu.

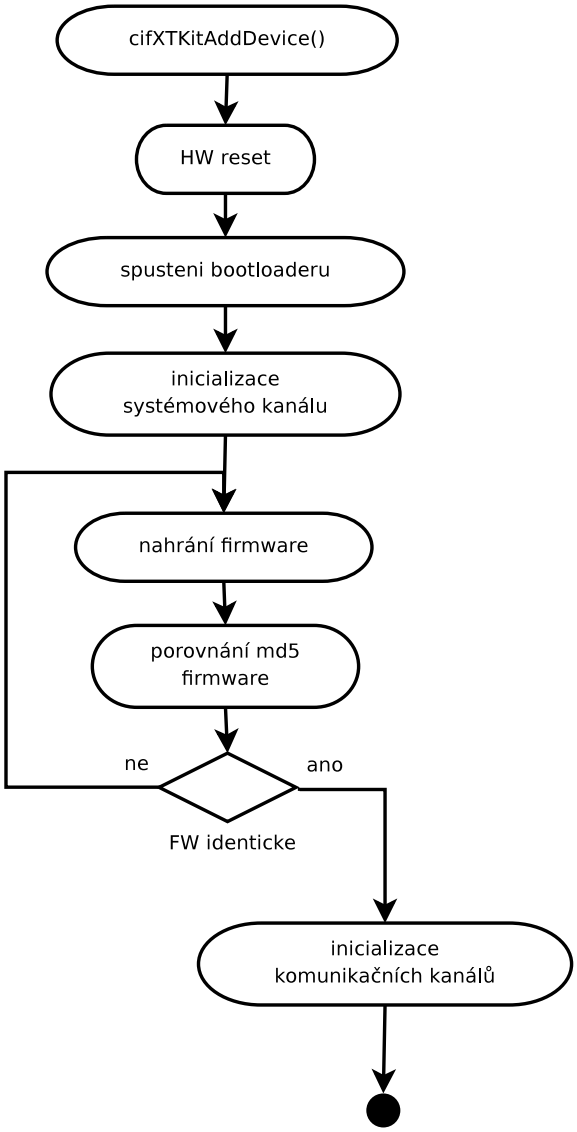
typ	název	popis
void*	pvDeviceInstance	ukazatel na nadřazenou strukturu DEVICEINSTANCE
unsigned char*	pbDPMChannelStart	začáteční adresa kanálu
unsigned long	ulDPMChannelLength	velikost kanálu v bajtech
unsigned long	ulChannelNumber	číslo komunikačního kanálu
int	fIsSysDevice	!=0 kanál je systémový
data schánek (mailboxů)		
NETX_SEND_MAILBOX_BLOCK*	ptSendMailboxStart	začáteční adresa odesílací schánky (send mailbox)
unsigned long	ulSendMailboxLength	velikost schránky v bajtech
NETX_RECV_MAILBOX_BLOCK*	ptRecvMailboxStart	začáteční adresa přijímací schránky (receive mailbox)
unsigned long	ulRecvMailboxLength	velikost schránky v bajtech
void*	ac_sync	synchronizační prvek pro příjem acyklických zpráv
char*	pPacket	buffer pro uložení přijatého paketu
unsigned long	ul_cmd_exp	při odeslání paketu, očekává příjem tohoto příkazu, jinak -1
void*	p_thread	identifikátor vlákna pro příjem paketů
uložená konfigurace		
void*	controller_info	informace o připojeném controlleru (je-li nějaký)
void*	device_info	informace o konfiguraci zařízení (zejména sloty/subsloty)
příznaky netXu		
unsigned short	usHostFlags	příznaky hosta (HSF nebo HCF)
unsigned short	usNetxFlags	příznaky netXu (NSF nebo NCF)
unsigned long	ulDeviceCOSFlags	příznaky změny stavu netXu
unsigned long	ulHostCOSFlags	příznaky změny stavu hosta
NETX_CONTROL_BLOCK*	ptControlBlock	ukazatel na kontrolní blok kanálu
synchronizační příznaky		
unsigned char	bHandshakeWidth	velikost synchronizační oblasti
NETX_HANDSHAKE_CELL*	ptHandshakeCell	synchronizační oblast kanálu
cyklická data		
PIOINSTANCE*	pptIOInputAreas	seznam oblastí vstupů
unsigned long	ulIOInputAreas	počet oblastí vstupů
PIOINSTANCE*	pptIOOutputAreas	seznam oblastí výstupů
unsigned long	ulIOOutputAreas	počet oblastí výstupů

Tabulka 5.2: Výčet prvků struktury CHANNELINSTANCE

5.3.1.2 Funkce hlavní části C-toolkitu

Zde budou popsány některé funkce definované v hlavní části C-toolkitu, výčet omezíme pouze na ty nejčastěji používané. Bližší popis viz A.3.0.3.

- `cifXTKitAddDevice()`
  Tato funkce je vstupním bodem pro inicializaci čipu netX. V rámci této funkce je do netXu nahrán firmware a ostatní inicializační rutiny, poté se inicializuje struktura DEVICEINSTANCE. Pro lepší popis je uveden vývojový diagram viz Obrázek 5.2. Po úspěšném dokončení této funkce je čip netX zinicializován a je možné zahájit inicializaci komunikačního stacku.



Obrázek 5.2: Vývojový diagram funkce `cifXTKitAddDevice()`

- `xChannelPutPacket()`
  Funkce sloužící pro acyklickou komunikaci ve směru **host** → **netX**. Veškerá acyklická volání od hosta jsou realizována touto funkcí. Tato funkce nakopíruje připravený paket do odesílací schránky vybraného kanálu a ošetří veškerou synchronizaci.

- `xChannelGetPacket()`
  Funkce sloužící pro acyklickou komunikace ve směru **netX** → **host**. Veškerá acyklická volání od netXu jsou realizována touto funkcí. Tato funkce nakopíruje paket z přijíamcí schránky do připraveného bufferu a ošetří veškerou synchronizaci.

- `xChannelIOWrite()`
  Funkce sloužící pro cyklickou komunikaci ve směru **host**→**netX**. Tato funkce nakopíruje data do zvolené oblasti obrazu výstupů a ošetří veškerou synchronizaci.

- `xChannelIORead()`
  Funkce sloužící pro cyklickou komunikaci ve směru **netX**→**host**. Tato funkce nakopíruje data do zvolené oblasti obrazu výstupů a ošetří veškerou synchronizaci.

5.3.2 Funkce specifické pro OS

Druhou částí je vrstva zapouzdřující operační systém. Jejím úkolem je poskytnout hlavní části toolkitu všechny potřebné funkce bez ohledu na použití OS. Tuto část bylo nutné napsat celou, i zde je uvedena kompletní. Při úpravě, nebo tvorbě nadřazené aplikace se totiž počítá se zde uvedenými funkcemi.

5.3.2.1 Správa paměti

Funkce pro práci s pamětí jsou prakticky identické, jako jsou použité v POSIXu. Jejich přepis je díky tomu velmi snadný. Pro všechny funkce postačují knihovny `stdlib.h` a `string.h`. Funkce jsou následující.

- `OS_Memalloc()` shodné s `malloc()`.
- `OS_Memrealloc()` shodné s `realloc()`.
- `OS_Memfree()` shodné s `free()`.
- `OS_Memset()` shodné s `memset()`.
- `OS_Memcpy()` shodné s `memcpy()`.

- `OS_Memcmp()` shodné s `memcmp()`.
- `OS_Memmove()` shodné s `memmove()`.

5.3.2.2 Správa času

Funkce pro správu času jsou v POSIXu odlišné, v rámci těchto funkcí je třeba provést úpravy. Veškeré použité funkce jsou definovány v knihovně `time.h`. Navíc tyto funkce vyžadují při kompilaci slinkování s knihovnou pro reálný čas (příznak `-lrt` při kompilaci).

- `OS_Sleep()`
  Používá volání `nanosleep()`, vstupní parametr (čas v milisekundách) je nutné přepočítat a naplnit strukturu `timespec`.
- `OS_GetMilliSecCounter()`
  Používá volání `clock_gettime()`, poté je ještě výstup přepočítán na milisekundy.

5.3.2.3 Operace s řetězci

Všechny funkce požadované v této části mají své ekvivalenty v knihovně `string.h`, přepis je díky tomu snadný.

- `OS_Strlen()` shodné s `strlen()`.
- `OS_Strncpy()` shodné s `strncpy()`.
- `OS_Strcmp()` shodné s `strcmp()`.
- `OS_Strnicmp` shodné s `strncasecmp()`.
- `OS_Strcat()` shodné s `strcat()`.
  Tato funkce byla přidána oproti výchozímu C-toolkitu, je použita v logovacích výpisech.
- `OS_Sprintf()` shodné s `sprintf()`.
  Také přidána, využívána při výpisech a také při plnění struktur.

5.3.2.4 Synchronizační objekty

Všechny synchronizační objekty jsou implementovány pomocí knihovny `pthread.h`. Navíc je vyžadováno slinkování s knihovnou (příznak `-lpthread` při kompilaci).

- `OS_CreateMutex()`
  Mutexy jsou implementovány standardně a jejich operace přesně odpovídají POSIXu. Aby bylo umožněno používat různé OS, je po volání `pthread_mutex_init()` vrácený ukazatel přetypován na `void*`.

- `OS_DeleteMutex()`
  Je shodné s `pthread_mutex_destroy()`.

- `OS_WaitMutex()`
  Čekání na mutex s timeoutem, implementováno jako `pthread_mutex_timedwait()`. Pro verzi knihovny `pthread` bez podpory tohoto volání je nutné použít opakované volání `pthread_mutex_trylock()` s čekáním. Nelze použít blokující volání `pthread_mutex_lock()`.

- `OS_ReleaseMutex()`
  Odemykání mutexu, stačí volání `pthread_mutex_unlock()`.

- `OS_CreateLock()`
  Zámky jsou implementovány pomocí spinlocků, toto zamykání musí být bezpečné proti signálům. Vytvoření zámku je provedeno voláním `pthread_spin_init()`. Ukazatel na zámek je stejně jako u mutexů přetypován na `void*`.

- `OS_DeleteLock()`
  Je použito `pthread_spin_destroy()`.

- `OS_EnterLock()`
  Použito `pthread_spin_lock()`, předtím je ještě zakázáno přijímání signálu SIGRTMIN+1 (resp. přerušení od netXu). Je jediné blokující volání, které je použito.

- `OS_LeaveLock()`
  Použito `pthread_spin_unlock()`, poté je opět povolen příjem signálu.

- `OS_CreateEvent()`
  Tento synchronizační objekt je implementován jako podmínková proměnná. Po volání `pthread_cond_init()` se jako u ostatních synchronizační objektů přetypovává na `void*`.

- `OS_DeleteEvent()`
  Je použito `pthread_cond_destroy()`.

- `OS_SetEvent()`
  Signalizuje příchod události, použito `pthread_cond_signal()`.

- `OS_ResetEvent()`
  Tato funkce je implementována jako prázdná, jelikož podmínkové proměnné z knihovny `pthread.h` se resetují automaticky.

- `OS_WaitEvent()`
  Čekání na příchod události, použito volání `pthread_cond_timedwait()`. Stejně jako u mutexů nesmí být blokující volání.

5.3.2.5 Operace se soubory

Ovladač přistupuje k souborům na pevném disku PC pouze za účelem načtení bootloaderu a firmware při inicializaci.

- `OS_FileOpen()`
  Otevření souboru, přístupová práva k souboru nejsou v hlavičce specifikována, soubor je otevřen s právy čtení a zápis s příznakem binárního přístupu.

- `OS_FileClose()`
  Pouze volá `fclose()`.

- `OS_FileRead()`
  Vyčte ze souboru požadovaný počet bajtů (čtení v binárním režimu), používá volání `fseek()` pro posun v souboru a `fread()` pro vlastní čtení.

- `OS_FileOpen_spec()`
  Tato funkce je přidána oproti výchozímu C-toolkitu, je použita v obslužných funkcích instalace (viz 5.3.3)

5.3.2.6 Obslužné rutiny pro PCI

Pro PCI kartu je nutné připravit rutiny pro čtení a zápis PCI registrů (popsáno již v 5.2.3.6). V user-space části ovladače se jedná o následující funkce.

- `OS_ReadPCIConfig()`
  Funkce která přes volání `ioctl()` vyčte do bufferu v user-space konfigurační registry PCI.

- `OS_WritePCIConfig()`
  Shodné s předchozí funkcí, pouze zápis místo čtení.

5.3.2.7 Mapování paměti

Ve výchozím C-toolkitu jsou připraveny dvě funkce pro mapování částí DPM do user-space. Tyto funkce byly zcela přepsány.

- `OS_MapUserPointer()`
  Mapování paměti do user-space, v původním C-toolkitu určeno pro mapování části paměti, ve stávající verzi již tato funkce nemá význam, protože DPM je namapována vždy celá. Tato funkce nyní volá `mmap()` a v rámci inicializace mapuje celou DPM netXu do user-space.

- `OS_UnmapUserPointer()`
  Ruší mapování paměti do user-space, stejně jako předchozí oproti původnímu C-toolkitu zcela změněno.

5.3.2.8 Obslužné rutiny přerušení

Tato část ovladače běží v user-space, přerušení zde proto nejsou k dispozici. V kernel modulu je při jejich zpracování poslán signál. Proto rutiny původně určené pro zpracování IRQ jsou přepsány na zpracování signálu.

- `OS_EnableInterrupts()`
  Tato funkce volá `ioctl()` a zaregistruje aplikaci v kernel modulu a při každém IRQ je aplikaci poslán signál na základě PID.

- `OS_DisableInterrupts()`
  Tato funkce také volá `ioctl()` ale ruší registraci v kernel modulu.

- `sig_handler()`
  Tato funkce je volána při přijetí signálu, volá DSR, ISR je již implementované v kernel modulu.

5.3.2.9 Inicializace a deinicializace

Ve výchozí verzi C-toolkitu jsou obě funkce definovány jako prázdné.

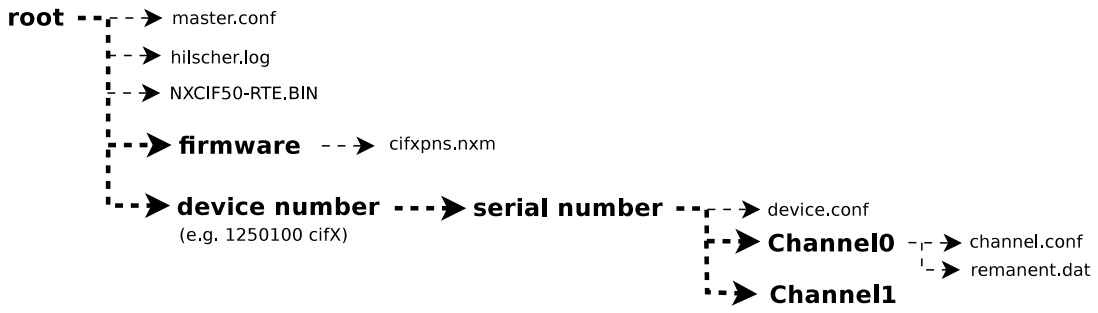
- `OS_Init()` V rámci této funkce inicializujeme globální proměnné a registrujeme obslužnou funkci signálu.
- `OS_Deinit()` Tato funkce je ponechána prázdná.

5.3.3 Obslužné funkce instalace

Poslední částí je vrstva zapouzdřující instalaci ovladače na PC. Tato část je zcela na volbě tvůrce ovladače.

5.3.3.1 Instalační struktura ovladače

Pro instalační strukturu ovladače nejsou žádné požadavky ani doporučení. Při použití ovladače pro Windows lze vyčíst strukturu z registrů. Pro Linux jsem strukturu prakticky zachoval, ovšem namísto registrů jsou použity textové soubory. Struktura instalační složky je na obrázku 5.3.



Obrázek 5.3: Struktura složky instalace

Tato struktura je použita se snahou logicky oddělit jednotlivé části.

V kořenovém adresáři jsou umístěny soubory pro libovolné zařízení s čipem netX. Ve složce `device number/serial number` jsou soubory použité pouze pro jedno zařízení a složky pro soubory jednotlivých komunikačních kanálů.

Pro data popisující konfiguraci zařízení jsem zvolil prosté textové soubory, pro snadnější rozpoznání jsou všechny s příponou `*.conf`. Tyto soubory lze upravovat ručně, případně pomocí obslužné aplikace viz A.4.2. Dat v těchto souborech není mnoho, proto

jsem zvolil velmi jednoduchou strukturu

klíč: hodnota

na každém řádku. Výchozí cesta k instalační složce je `/usr/local/hilscher`. Aby bylo možné použít i jinou, jsou na soubory `master.conf` a `hilscher.log` ještě vytvořeny symbolické odkazy ve složce `/etc/hilscher`.

Následuje popis jednotlivých složek a souborů.

`./`

Výchozí složka instalace, zde uváděna jako kořenový adresář.

- `master.conf`
  Tento soubor v sobě obsahuje cestu do kořenového adresáře instalace. Výchozí cesta je `/usr/local/hilscher`. Dále obsahuje cestu k souborům firmware.
- `hilscher.log` Tento soubor slouží pro ukládání ladících výpisů.
- `NXCIF50-RTE.BIN` Soubor bootloaderu.

`device_number/serial_number`

V této složce jsou umístěny soubory týkající se pouze jednoho zařízení.

- `device.conf`
  Tento soubor obsahuje alias karty cifX na fieldbusu a určení, jestli se mají používat přerušení.

`device_number/serial_number/Channel0`

V této složce jsou soubory týkající se pouze jednoho komunikační kanálu, typicky jeden až dva.

- `channel.conf`
  Tento soubor určuje, který firmware se má pro komunikační kanál použít.
- `remanent.dat`
  Soubor obsahující nastavení inicializace zařízení IO device, jsou-li parametry (alias, IP....) přijaty od controlleru s příznakem remanent.

`firmware`

Složka obsahující veškeré soubory firmware, její název lze změnit pokud opravíme i cestu v souboru `master.conf`.

Je vytvořen instalační skript, který celou instalační strukturu vytvoří. Dále je připravena aplikace, ve které lze naplnit konfigurační soubory. Není tedy nutné jakoukoli část měnit ručně. Všechny aplikace jsou v příloze viz A.

Jednotlivé funkce zapouzdřující instalační strukturu jsou popsány v A.3.0.3. Jedná se o soubor `tKitUser.c` a všechny funkce lze snadno rozlišit podle předpony `USER_`.

5.4 C-toolkit pro powerPC

C-toolkit pro variantu, kde hostem je powerPC MPC5200 a je použit modul comX je nutné celý toolkit upravit oproti verzi pro PC s kartou cifX. V této kapitole budou popsány právě tyto úpravy, jinak se budeme odkazovat na kapitolu 5.3 C-toolkit pro PC.

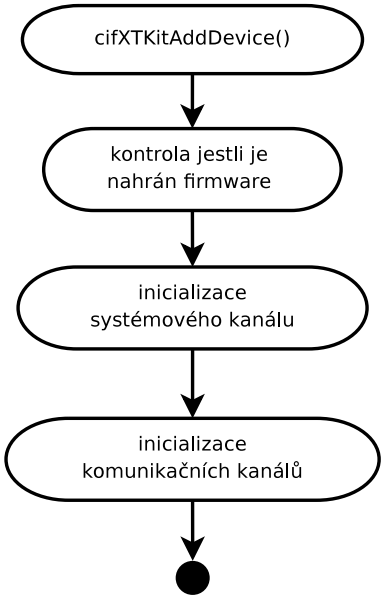
5.4.1 Hlavní část

Obsluha modulů cifX a comX má několik odlišností, které se promítají do celého ovladače. V hlavní části C-toolkitu jsou to zejména tři níže uvededné rozdíly, co zde není uvedeno je totožné s C-toolkitem pro PC.

5.4.1.1 Změna správy firmware

Modul comX řeší nahrávání firmware o poznání elegantněji než karta cifX. Do modulu stačí nahrát firmware pouze jednou a ne při každém spuštění ovladače. Tím je možné ušetřit paměť na straně host CPU, ktrou by zabíral firmware. Ten navíc dosahuje velikosti až 1MB. Dále se tím výrazně zrychlí inicializace netXu. Z C-toolkitu pro powerPC jsem proto zcela odstranil všechny funkce spravující nahrávání firmware, jedná se o prakticky celý soubor `cifXDownload.c` a části souboru `cifXInit.c`. Veškerá správa je přesunuta do obslužné aplikace viz A.4.1.

Vývojový diagram pro funkci `cifXTKitAddDevice()` se zjednoduší viz Obrázek 5.4. Z něj je mimo jiné vidět, že toolkit nekontroluje, zda firmware je správný. Očekává, že toto provede uživatel pomocí obslužné aplikace pro firmware (viz A.4.1).



Obrázek 5.4: Vývojový diagram pro `cifXTKitAddDevice()` na powerPC

Dále změna správy firmware umožňuje výrazně omezit nutnost práce se systémem souborů, toto je popsáno v kapitole Obslužné funkce instalace (viz 5.4.3).

5.4.1.2 Pořadí bajtů (byte order)

Čip netX obsahuje jádro typu ARM, které používá little endian, host MPC5200 používá architekturu big endian. I přesto že při konfiguraci sběrnice LocalPlus Bus používáme protočení pořadí bajtů je nutné u všech dat, která prochází mezi hostem a netXem měnit pořadí bajtů.

Pro toto jsou připravena dvě makra.

- `LONG_ENDIAN()` pro 32 bitové proměnné
- `SHORT_ENDIAN()` pro 16 bitové proměnné

Tato makra je nutné připsat do všech funkcí, které zpracovávají data s netXu nebo plní pakety před odesláním.

5.4.1.3 Změna přístupu do DPM

Modul comX je k MPC5200 připojen sběrnicí LocalPlus Bus (viz [8]) s pevnou šířkou. Každý zápis i čtení z DPM probíhá vždy po 16bitech dat. Nelze tedy vyčíst nebo zapsat

jeden bajt. Dále adresy jsou také vždy 16bitové a při čtení od liché adresy přijdou neplatná data. Tento problém si vyžádal zcela předělat přístup do DPM. Funkce `memcpy()` původně použitá v toolkitu pro PC nelze použít. Je nutné všechny přístupy do DPM v celém toolkitu přesměrovat na novou funkci, která bere ohled na požadavky a přistupuje do DPM správně. Těmito funkce jsou `OS_read_DPM()` a `OS_write_DPM()` (popis viz 5.4.2).

5.4.2 Funkce specifické pro OS

Úpravy v hlavní části toolkitu uvedené výše si vyžádaly přidání nových funkcí do vrstvy zapouzdřující OS. Obsah funkcí popsaných v 5.3.2 je nezměněn.

5.4.2.1 Funkce pro přístup do DPM

Většina funkcí je shodná jako u C-toolkitu pro PC (viz 5.3.2). Jediné úpravy si vyžádala změna přístupu do DPM

- `OS_read_byte()`
  Tato funkce vyčte právě jeden bajt z DPM. Funkce nejprve zaokrouhlí danou adresu na nejbližší nižší sudou, poté vyčte jedno slovo (16 bitů) a vrátí vyšší nebo nižší bajt podle požadavku.

- `OS_write_byte()`
  Tato funkce zapíše právě jeden bajt do DPM. Zde je nutné nezměnit obsah ostatních adres. Funkce stejně jako `OS_read_byte()` nejprve zaokrouhlí adresu na nejbližší nižší sudou, poté vyčte jedno slovo, změní požadovaný bajt a celé slovo zapíše zpět.

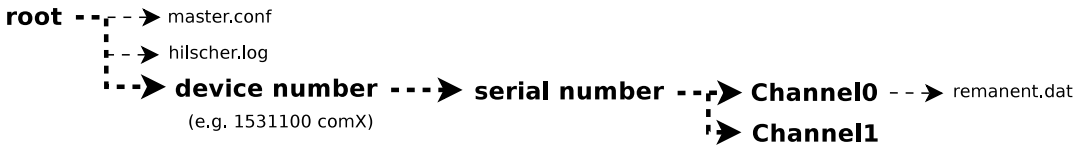
- `OS_read_DPM()`
  Funkce, která naplní připravený buffer daty z DPM od zadané adresy. Je ošetřeno čtení z lichých adres stejně jako u funkce `OS_read_byte()` a to jak na začátku tak na konci rozsahu adres.

- `OS_write_DPM()`
  Funkce, která zapíše připravený buffer na zadanou adresu do DPM. Zápis na liché adresy je ošetřen stejně jako u funkce `OS_write_byte()` a to jak na začátku tak na konci rozsahu adres.

5.4.3 Obslužné funkce instalace

Díky změně správy firmware je možné tuto část výrazně omezit. Z uživatelských funkcí definovaných pro C-toolkit na PC zůstane jen logování.

Instalační struktura je zachována shodná jako u C-toolkitu pro PC. Je tam však uloženo méně souborů. Jmenovitě není na systému souborů hosta uložen firmware, ten se nahrává jen jednou a poté jedině při aktualizaci firmware. Dále není potřeba soubor bootloaderu (NXCIF50-RTE.BIN), protože modul comX má v sobě bootloaderu nahrán napevno a není třeba jej nahrávat. Nejsou třeba ani konfigurační soubory pro zařízení a kanály. Pro zařízení stačí jen soubory `master.conf`, `hilscher.log` a `remanent.dat`.



Obrázek 5.5: Struktura složky instalace

5.5 DPM API

DPM API označuje nejvyšší vrstvu ovladače. Tato implementuje funkce přímo nabízené uživatelské aplikaci pro práci na sběrnici PROFINET.

Tato vrstva je závislá na použitém protokolu, ne na typu host CPU, je proto zcela shodná pro zařízení na PC i powerPC.

5.5.1 Část nezávislá na protokolu

Tuto část lze použít pro všechny komunikační protokoly. Čip netX umožňuje mnoho funkcí implementovat nezávislé na protokolu. Vše co šlo je implementováno v této části. Jedná se typicky o inicializaci, cyklická data a správu netXu.

Dále je implementováno několik menších pomocných funkcí, více viz A.3.0.4.

5.5.1.1 Inicializace, deinicializace

Zde jsou uvedeny všechny funkce pro inicializaci netXu (ne stacku) a přidružené kroky. Jediné, co nelze implementovat pro komunikační protokoly jednotně je acyklická komunikace. Tedy nastavení vlastního stacku a všechny acyklické služby PROFINETu je nutné implementovat pouze pro jeden protokol, což je popsáno v 5.5.2.

- `nX_init()`
  Funkce pro inicializaci netXu od nuly. Tuto funkci stačí volat jen jednou ihned po spuštění ovladače, poté již není použita. V rámci této funkce se inicializují globální proměnné, alokuje se paměť a plní se část struktury DEVICEINSTANCE. Nakonec se volá funkce `cifXTKitAddDevice()`. Po úspěšném ukončení této funkce je zařízení připraveno, ale stále ještě nekomunikuje po fieldbusu.

- `nX_channel_open()`
  Tato funkce zpřístupní strukturu `CHANNELINSTANCE` pro zvolený kanál připravenou v rámci funkce `cifXTKitAddDevice()`. Po tomto volání lze začít s kanálem pracovat, stále však není povolena komunikace po fieldbusu.

- `nX_channel_open_com()`
  Tato funkce povoluje komunikaci po fieldbusu, prakticky nastavuje bit `APP_COS_BUS_ON` v registru Příznaky změn stavu hosta (viz 3.4.2.6). Po jejím dokončení je netX připraven na práci na fieldbusu. Toto volání je nutné volat až po dokončení inicializace komunikačního stacku.

- `nX_close_com()`
  Tato funkce ukončí komunikaci netXu na fieldbusu.

- `nX_close()`
  Tato funkce uzavře kanál, poté s ním již nelze nijak pracovat, jedině znovu provést celou inicializaci.

- `nX_deinit()`
  Tato funkce slouží k ukončení ovladače dále vyčištění a uvolnění paměti.

- `nX_reset()`
  Tato funkce slouží k zahození všech nastavení stacku a umožňuje jeho novou inicializaci. Při tomto volání je restartován celý netX včetně OS. Veškerá komunikace po fieldbusu na všech kanálech se přeruší bez ohledu na okamžitý stav.

5.5.1.2 Cyklická komunikace

Odesílání a příjem cyklických dat probíhá nezávisle na zvoleném protokolu. V rámci těchto funkcí stačí pouze nahrát (alt. vyčíst) data z oblasti cyklických dat v DPM (viz 3.4.1).

- `nX_send_IO()`
  Funkce pouze zapíše data do zvolené oblasti DPM. Vlastní zápis probíhá přes funkci `OS_write_DPM()` (viz 5.4.2).

- `nX_read_IO()`
  Funkce pouze vyčte data ze zvolené oblasti DPM. Vlastní čtení probíhá přes funkci `OS_read_DPM()` (viz 5.4.2).

5.5.1.3 Diagnostické funkce

Jako poslední skupina funkcí zde budou uvedeny funkce sloužící k diagnostice, jedná se o výpisy stavu pro uživatele.

- `nX_dump_regs()`
  Tato funkce slouží pro vyčtení aktuálních hodnot registrů popsaných v 3.4.2. Toto se hodí zejména při spouštění zařízení, lze takto snadno vyčíst aktuální stav.

- `get_com_sta()`
  Tato funkce pouze vrací informaci, zda je aktivní komunikace po fieldbusu.

- `dbg()`
  Funkce sloužící k debuggovacím výpisům, výpisy lze přesměrovat buď na standardní výstup (terminál), nebo do souboru.

5.5.2 Část pro PROFINET

Tato část popisuje acyklickou komunikaci mezi host CPU a čipem netX. Ta jediná je zcela závislá na protokolu. V rámci této práce bylo implementováno rozhraní pro PROFINET IO device. Popis rozhraní a práce s ním již byl uveden viz 3.5.

Zde budou popsány pouze nejdůležitější funkce, kompletní popis viz A.3.0.5.

5.5.2.1 Datové struktury

Nejprve popíšeme datové struktury, které jsou použity pro uchování informací o zařízení (device) i nadřazeném controlleru. Kompletní výčet prvků viz A.3.0.5.

- `PNC_info_t`
  Struktura sloužící k uchování informací o controlleru. Slouží zejména pro diagnostiku.

- `PNS_info_t`
  Struktura k uchování informací o zařízení. Je zde uložena kompletní konfigurace stacku, konfigurace modulů i submodulů. Dále obsahuje strukturu pro uložení pevné konfigurace (viz níže).

- `PNS_non_volatile_setting_t`
  Tato struktura v sobě obsahuje veškerá nastavení, která jsou pevná (non-volatile), jedná se o alias a nastavení sítě. Při deinicializaci ovladače je uložena do souboru a při inicializaci načtena. Jsou-li v ní nějaká nastavení, použijí se přednostně. K vymazání lze použít funkci `reset_to_default()`.

5.5.2.2 Inicializace, deinicializace

Popis inicializace je uveden v kapitole 3.5.1. Zde popsané funkce pouze provádějí tyto kroky.

- `manual_start()`
  Funkce sloužící k zjednodušení volání při inicializaci. Nejprve vyčte pevné nastavení, je-li nějaké. Poté postupně volá funkce pro nastavení stacku. Jde jmenovitě o kroky již popsané v 3.5.1.1, v rámci programu se jedná o následující funkce.

  - `register_ap()`
    Tato funkce realizuje registraci příjmu acyklických zpráv.
  - `set_network_conf()`
    Je-li nastavení sítě pevně předdefinováno, je tímto voláním netXu přiřazeno. Při přeskočení tohoto volání je síťové nastavení získáno přes DHCP.
  - `set_dev_info()`
    Tímto voláním nastavíme informaci o zařízení.
  - `open_device()`
    Tímto voláním se realizuje naplnění bloku I&M zařízení.
  - `add_api()`
    Tato funkce realizuje vložení API.
  - `plug_init()`
    V rámci této funkce jsou připojeny systémové moduly a submoduly do slotu 0.

  - `channel_init()`
    Tímto voláním fakticky nastavíme stack všemi parametry uvedenými u výše zmíněných funkcí.

- `get_dev_handle()`
  Toto volání vrácí identifikátor, který slouží pro acyklická volání jako například připojování modulů.

- `plug_module()`
  Funkce pro připojení jednoho modulu. Před připojením musí být již ukončeno volání `manual_start()`.

- `plug_submodule()`
  Funkce pro připojení jednoho submodulu. Modul musí být již připojen.

- `pull_module()`
  Funkce pro odpojení modulu. Tímto voláním se zruší i všechny submoduly tohoto modulu.

- `pull_submodule()`
  Funkce pro odpojení jednoho submodulu.

- `warm_start()`
  Alternativní možnost konfigurace stacku. Použije se inicializace pomocí warmstart paketu. Tato varianta není doporučena, protože při ní nelze zjistit, které moduly jsou použity viz 3.5.1.2.

- `reset_to_default()`
  Touto funkcí se zruší veškeré předchozí (i pevné) nastavení a stack je nově inicializován. Zařízení má alias prázdný řetězec, nemá pevné síťové nastavení. Moduly a submoduly jsou zachovány, resp jsou odebrány a poté opět připojeny v identické konfiguraci.

- `PN_init()` a `PN_deinit()`
  Tyto funkce slouží k připravení (alt. zrušení) struktury `PNS_info_t` a `PNC_info_t` (viz 5.5.2.1). Obě tyto struktury slouží k uchování informací o controlleru a zařízení (stav a konfigurace).

5.5.2.3 Komunikace controller ↔ device

Zde uvedená volání zpracovávají acyklické zprávy PROFINETu. Tato volání se již netýkají inicializace.

- `alarm_diagnostic()`
  Slouží k odeslání diagnostického alarmu controlleru. Po odeslání funkce čeká na potvrzení, pokud nepřijde je vrácen chybový kód.

- `alarm_process()`
  Slouží k odeslání procesního alarmu, mechanizmus je shodný jako u diagnostického alarmu.

- `alarm_ret_submodule()`
  Slouží k odeslání alarmu návrat submodulu.

- `process_indication()`
  Všechny ostatní uvedená volání jsou zahajována aplikací v host CPU a čekají na odpověď. Tato funkce slouží jako vstupní bod při příjmu zpráv generovaných netXem. Tyto zprávy jsou buď iniciovány controllerem nebo netXem. Z přijatého paketu vyčteme příkaz (`ulCmd` viz 3.4.3.3) a podle něj voláme funkci pro zpracování. Tyto funkce jsou uvedeny níže, nejsou přístupné pro vyšší vrstvu (tj. uvedeny v `PN_api.h`).

  - `write_req_res()`
    Funkce pro zpracování paketu Write request (viz 2.1.1).
  - `read_req_res()`
    Funkce pro zpracování paketu Read request. Tato funkce je doplňková k předchozí (`write_req_res()`), zde controller pouze čte parametrizační data.
  - `reset_to_default()`
    Tato funkce je jediná přístupná pro nadřazenou aplikaci. Její popis je již uveden vyše.
  - `controller_COS()`
    Funkce pro zpracování změn stavu controlleru. Ve stávající verzi jen vypíše do konzole změnu.
  - `fill_controller_info()`
    Funkce, která při inicializaci uloží informace o controlleru.

- `print_conf_dif()`
  Dojde-li při inicializaci k nalezení rozdílu v konfiguraci IO device s konfigurací, kterou očekává controller, je toto indikováno. Tato funkce zatím jen vypisuje rozdíl v konfiguraci. Pozn. Není indikována absence modulu/submodulu, jen pokud je připojen jiný.
- `save_ip()`
  Při inicializaci bez pevného síťového nastavení je použito DHCP, po obdržení adresy se zavolá tato funkce, která nastavení uloží. Je-li nastavení požadováno jako pevné, je uloženo do struktury `PNS_non_volatile_setting_t`.
- `save_name()`
  Shodná funkce jako předchozí, jen ukládá jméno zařízení (alias) získané od controlleru.
- `save_type()`
  Shodné s předchozí, jen ukládá typ zařízení.
- `send_short_ack()`
  Funkce pro obsluhu paketů, u kterých není třeba žádné zpracování jen potvrzení příjmu. Funkce pouze odpovídá na paket krátkým potvrzením.

5.5.2.4 Diagnostika

Níže uvedené funkce slouží uživateli, případně host CPU k zjištění stavu netXu alt. stacku. Nemají žádný vliv na komunikaci po fieldbusu.

- `get_diagnostic()`
  Funkce, která vrací diagnostický buffer.
- `get_station_name()`
  Funkce, která vrací aktuální alias zařízení na fieldbusu (např. "comxrepns").
- `get_station_type()`
  Funkce, která vrací aktuální typ zařízení (např. "pnios").
- `get_network_conf()`
  Funkce, která vrací aktuální síťové nastavení komunikačního portu.
- `get_modules_layout()`
  Pokud při zapojení modulů dojde k chybě, lze tímto voláním zjistit které moduly-/submoduly jsou přítomny. Toto volání nelze použít při již probíhající komunikaci.

Funkce používá princip plug/pull s kontrolou statusu, při probíhající komunikaci je tato přerušena. Pokud je použita manuální konfigurace, není toto volání doporučeno, v tom případě je doporučeno vyčíst konfiguraci z `PNS_info_t`.

- `print_info()`
  Funkce vypisující informace o zařízení (vypisuje struktury `PNS_info_t` a `PNC_info_t`).

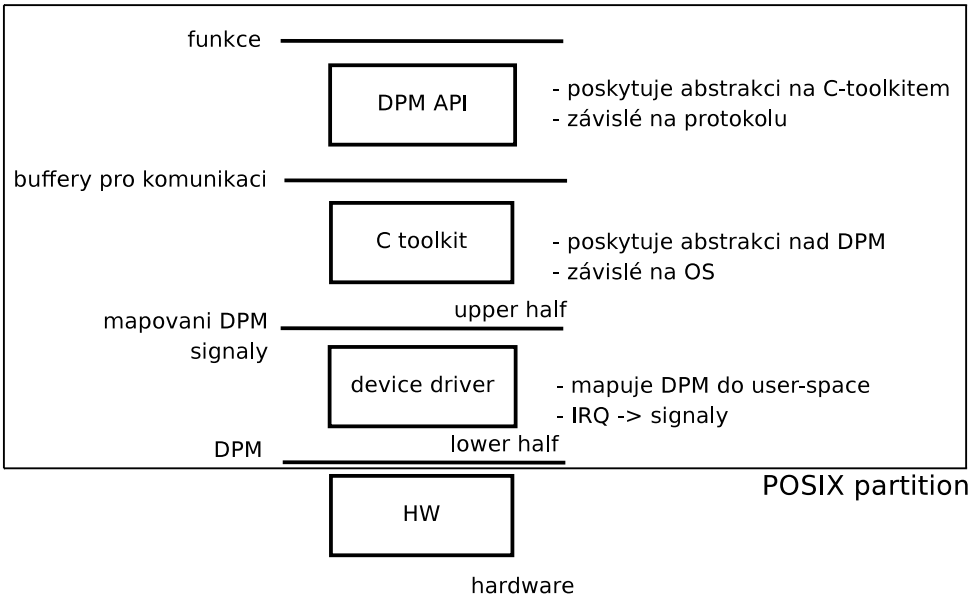
Kapitola 6

Ovladač pro PikeOS

V této kapitole bude popsán ovladač vyvinutý pro OS PikeOS. PikeOS byl použit pouze pro host powerPC MPC5200. Mnohé části jsou podobné jako pro ovladač pro OS Linux, proto se budeme odkazovat na kapitolu 5.

6.1 Struktura ovladače

Struktura ovladače je podobná jako pro OS Linux, rozdíly vyplývají pouze ze specifikace PikeOSu, tyto budou popsány níže.



Obrázek 6.1: Struktura ovladače pro PikeOS

Celá aplikace běží v POSIXovém oddílu (POSIX partition). Toto řešení je nejvýhodnější s ohledem na přenositelnost kódu mezi verzí pro Linux. V rámci tohoto oddílu navíc celá aplikace sdílí adresní prostor, díky tomu lze ušetřit všechny funkce vyžadované v Linuxu pro komunikaci mezi kernel modulem a částí v user-space.

Pro PikeOS je třeba vytvořit tzv. integrační projekt, který určuje nastavení celého OS. Ten je uveden v příloze A.1 ve složce `src/comX_PikeOS/dva_int`. Popis jak vytvořit integrační projekt je uveden například v [21].

PikeOS na rozdíl od Linuxu nepoužívá rozdělení na kernel a user-space, ale používá ovladače rozdělené na tzv. lower a upper half (viz [12]).

Nejvýraznějším omezením v PikeOSu je absence systému souborů s možností zápisu (viz [12]). PikeOS používá systém souborů pouze pro čtení (ROM). Z tohoto vyplývají určitá omezení, popsána níže.

6.2 Ovladač zařízení

Pro PikeOS je nutné kompletně přepsat kernel modul popsaný v 5.2. Zde se nepoužívá pojem kernel modul ale ovladač zařízení (device driver, nebo lower half) viz [12]. Ten funkcí nahrazuje kernel modul. Dále s ním má společné to, že je psán v samostatném souboru a má jiná oprávnění než aplikace v POSIXu.

V ovladači se řeší zejména inicializace komunikace po sběrnici LocalPlus Bus.

6.2.1 Zavedení ovladače

Zavedení ovladače v PikeOS se provádí pomocí funkce `dd_install_driver()`, kterou zavolá POSIXová aplikace. V rámci zavední se provádějí následující kroky.

- Alokování paměti
- Namapování paměti
  Mapuje se oblast registrů MPC5200. Fyzická adresa se vyčítá pomocí tzv. properties (viz [13]). Tímto lze měnit fyzickou adresu v rámci integračního projektu a není nutné zasahovat do zdrojového kódu ovladače.

Ovladač se zavádí při každém spuštění ovladače (tj. při každém spuštění POSIXového oddílu). Zavedení ovladače (`dd_install_driver()`) proto prakticky splývá s jeho otevřením (funkce `open()`).

V rámci otevření ovladače se provede zbývající inicializace.

- Namapování DPM
  Mechanizmus je shodný jako u mapování registrů MPC5200 (viz výše).

- Nastavení periférií procesoru
  Toto je shodné z funkcí `open()` u kernel modulu pro Linux (viz 5.2.4.2).

6.3 C-toolkit

C-toolkit pro PikeOS vychází z verze pro Linux na powerPC. Rozdělené je shodné (viz 5.4). Pro každou část zde bude uveden popis co je rozdílné od ovladače pro Linux, co zde není uvedeno je shodné.

6.3.1 Hlavní část

Tato část je totožná jako ve verzi ovladače pro Linux uvedené v 5.4.1.

6.3.2 Funkce specifické pro OS

POSIXové rozhraní PikeOS v2.2 je odlišné od rozhraní v Linuxu, toto si vyžádalo několik změn ve funkcích zapouzdřujících funkce OS.

6.3.2.1 Synchronizační objekty

Všechny synchronizační objekty jsou implementovány pomocí knihovny `pthread.h`.

- `OS_CreateLock()`
  V PikeOSu jsou jako zámky použity mutexy, spinlocky nejsou k dispozici. Pro použití musí splňovat požadavky uvedené v 5.3.2.4. Tato funkce je přesměrována na `OS_CreateMutex()`.

- `OS_DeleteLock()`
  Funkce je přesměrována na `OS_DeleteMutex()`.

- `OS_EnterLock()`
  Použito `pthread_mutex_lock()`, předtím je ještě zakázáno přijímání signálu SIGRTMIN+1 (resp. přerušení od netXu). Je jediné blokující volání, které je použito.

- `OS_LeaveLock()`
  Použito `pthread_mutex_unlock()`, poté je opět povolen příjem signálu.

- `OS_WaitMutex()`
  POSIX pro PikeOS nepodporuje volání `pthread_mutex_timedwait()`, proto je nutné funkci implementovat jako opakované volání funkce `pthread_mutex_trylock()` s čekáním (tři pokusy). Nelze použít blokující volání `pthread_mutex_lock()`.

6.3.2.2 Operace se soubory

Vzhledem k absenci systému souborů s možností zápisu je tato skupina funkcí zcela vynechána. Pro PikeOS nemá žádný význam.

6.3.2.3 Mapování paměti

Celý systém mapování paměti je odlišný od Linuxu. DPM je mapována již při inicializaci ovladače a není nutné ji volat poté. Obě funkce pro mapování paměti jsou ponechány prázdné.

6.3.3 Obslužné funkce instalace

Jak bylo již výše uvedeno, PikeOS neposkytuje systém souborů s možností zápisu (pouze čtení). Instalační adresář tedy nelze realizovat. Z obslužných funkcí instalace zbude pouze funkce pro logování navíc přesměrována ze souboru `hilscher.log` na standardní výstup.

6.4 DPM API

DPM API je prakticky identické jako v ovladači pro Linux (viz 5.5).

Rozdíl je v inicializační funkci `nX_init()`, kde je rozdílné volání při mapování paměti. Činnost, která se vykonává v rámci funkce je však identická.

Dále je drobný rozdíl ve vláknové funkci čekající na nově příchozí pakety. Zde je třeba při přijetí paketu vždy odemknout mutex. Při použití funkce `OS_ReleaseMutex()` se mutex neodemkne, tento problém byl obejit a ve vlákně se volá funkce `OS_InitMutex()`, která jej nově zinicializuje do odemčeného stavu.

Kapitola 7

Testování zařízení

Zařízení bylo testováno na funkčnost komunikace po PROFINETu pomocí běžného komerčního PROFINET IO controlleru a aplikace pro testování zařízení PROFINET IO device. Dále se testovalo zatížení procesoru ovladačem. Všechny testy jsou uvedeny níže.

7.0.1 Wireshark

Pro obě varianty testování je třeba sledovat pakety na síti. Toto je v obou případech realizováno pomocí programu Wireshark (viz [15]). Tento program patří mezi nejběžnější programy pro sledování sítí. Dokáže navíc rozkódovat pakety protokolu PROFINET, lze tedy vyčítat data bez potřeby nahlížení do specifikace PROFINETu pro určení hodnot. Toto je výhodné zejména pro větší pakety jako například Connect request (ten má zhruba 900kB). Dále je možné pakety filtrovat, výhodné je zejména potlačit zobrazení cyklických dat. Toto lze provést například filtrem.

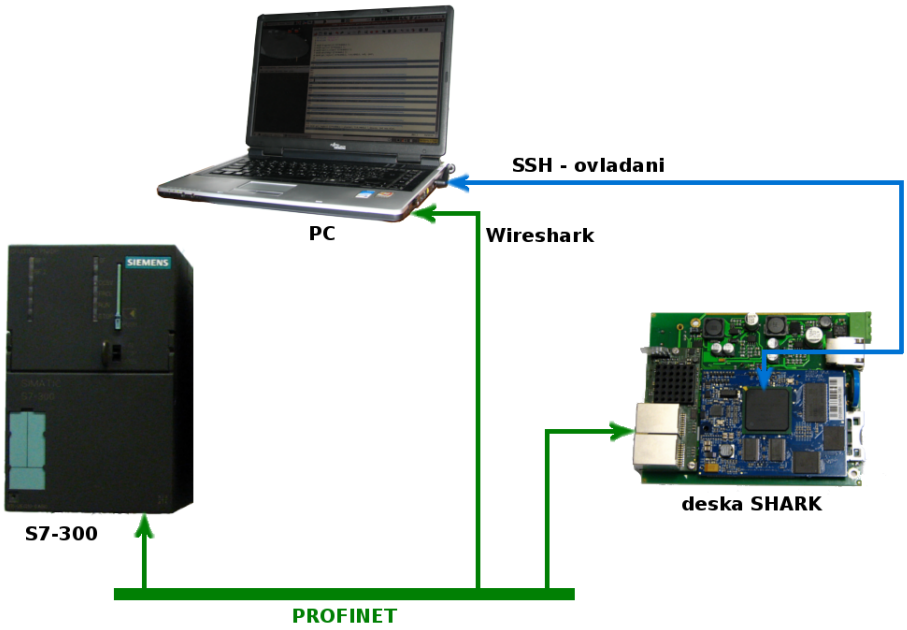
```
!(pn_rt.frame_id == 0xC080 || pn_rt.frame_id == 0xC010)
```

Sled zachycených paketů se ukládá do souboru `*.pcap`.

7.1 Testování s controllerem S7-300

Zařízení bylo od začátku vývoje zkoušeno s controllerem S7-300. Tento controller je typický zástupce komerčně dostupných zařízení. Výrobce je firma Siemens, která je také autorem normy PROFINET. Použitý controller byl S7-315PN/DP v2.3. Pro všechna zařízení byla pomocí něj testována základní funkčnost komunikace, zejména inicializace.

Pro potřeby testování, kde chceme sledovat i všechny pakety na síti je třeba zapojení podle Obrázku 7.1. Připojení počítače pro odchyt paketů je nutné realizovat pomocí switche s možností zrcadlení portů. Je potřeba jeden komunikační port (buď od S7-300 nebo netXu) nazrcadlit do PC.



Obrázek 7.1: Schéma zapojení pro testování s S7-300

Všechny varianty zařízení, tedy karta cifX na OS Linux a modul comX na OS Linux a PikeOS, úspěšně komunikují s controllerem S7-300. Z pohledu controlleru se jeví jako identické.

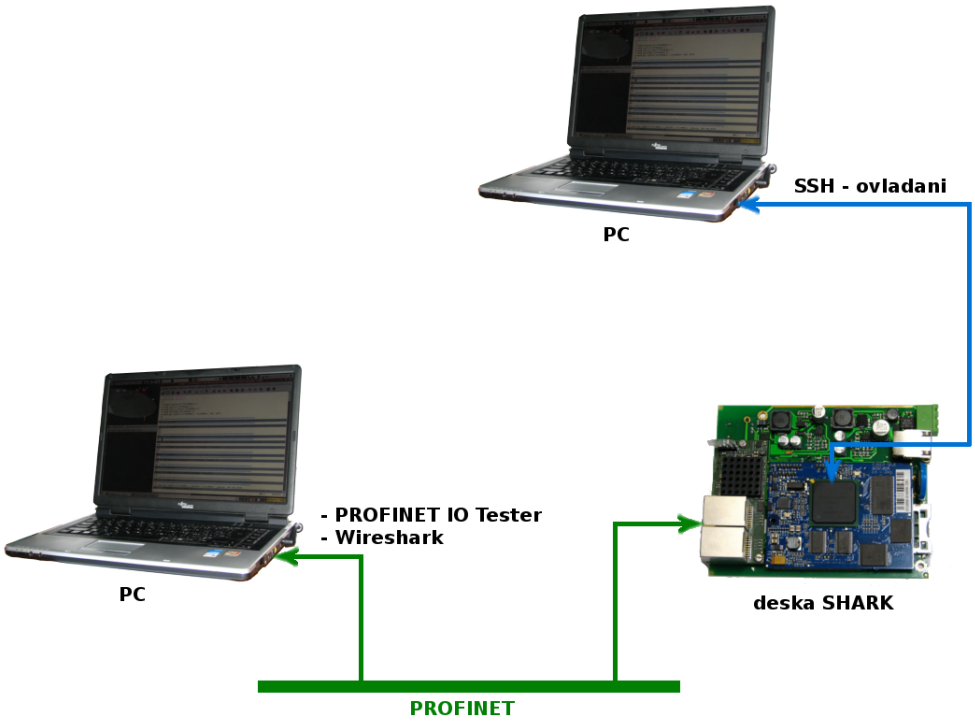
Typický průběh komunikace z pohledu uživatele je následující. Po spuštění controller okamžitě zahajuje inicializaci PROFINETu. NetX jako device zatím neaktivní, teprve nabíhá host CPU. Na controlleru po skončení jeho vnitřní incializace svítí BF (bus fault) a SF (chyba, nenašel netX podle HW konfigurace). Na čipu netX svítí červená a zelená LED (zelená pro OS netXu, červená pro BF). Poté inicializujeme ovladač a čip netX (viz 3.5.1.1). Poté je zahájena komunikace po PROFINETu. Nejprve se rozblikají obě LED BF a po dokončení připojení obě zhasnou a probíhá komunikace.

Logy z testování jsou přiloženy v příloze (viz A.5.1), vždy log z host CPU (soubor *.log a sled paketů v *.pcap).

7.2 Testování PROFINET IO Testerem

Po odladění zařízení pomocí controlleru S7-300 jsem ještě zařízení testoval pomocí aplikace PROFINET IO Tester (viz [16]). Tato aplikace slouží ke kompletnímu testování zařízení PROFINET IO device. Zahrnuje řadu testovacích případů, které jsou shodné s akceptačními testy v laboratořích PI. Tyto testy jsou nejlepším ukazatelem správnosti funkce zařízení. Program využívá běžnou síťovou kartu PC pro emulaci chování PROFINET IO controlleru, není tedy třeba žádný HW navíc.

Při testování bylo použito zapojení podle Obrázku 7.2. Zde není třeba žádný switch ani jiný síťový prvek.



Obrázek 7.2: Schéma zapojení pro testování s PROFINET IO testerem

Použitá verze programu byla `PROFINET_IO_Tester_1_5_Revision_4282` s testovacím protokolem pro PROFINET `PNIO_V2_2_2_14_15_PUB`.

Pro každý testovací případ je uveden stručný výsledek testu a to formou obrázku na pravé straně, kompletní log je uložen v příloze viz A.5.2. Pro každý případ je uveden log z programu PROFINET IO tester (ve formátu html) dále log z konzole host CPU (v textovém souboru *.log) a sled paketů z programu Wireshark (soubory *.pcap).

Program PROFINET IO Tester je napsaný pouze pro Windows. Po registraci na stránkách výrobce (viz [16]) lze stáhnout instalaci. Jako první je nutné nainstalovat program WinPcap (verze 3.1), tím se vytvoří v záložce nastavení sítě nový protokol a to Network Monitor Driver. Dále je třeba mít nainstalován .NET verze 3.5. Při instalaci je třeba vyplnit číslo, které vám přijde mailem, poté máte 14 dní plně funkční program. Program nepotřebuje připojení na internet, ani při registraci ani při práci. Po vypršení této doby se program omezí na demo, které již nelze pro testování použít. Program je pro delší používání potřeba zakoupit, lze i opakovaně přeinstalovávat počítač (nejsnáze přes záložní obraz disku).

Při testování lze na stejném počítači spustit program Wireshark a přímo sledovat pakety na síti, není k tomu potřeba žádný HW navíc. Při instalaci Wiresharku je třeba zachovat WinPcap verze 3.1, doporučené pořadí instalace je následující. Nejprve WinPcap, poté Wireshark se zachováním WinPcap (Wireshark obsahuje novější verzi 4.0 se kterou PROFINET IO Tester nefunguje) a nakonec PROFINET IO Tester.

Při práci s programem je nutné zakázat všechny protokoly pro zvolenou síťovou kartu, kromě Network Monitor Driveru. Poté založíme projekt a můžeme testovat, podrobný popis tohoto je uveden v [17]. K programu je přiložen testovací soubor pro PROFINET a to `PNIO_V2_2_2_14_15_PUB`.

U každého testovacího případu jsou čtyři možnosti výsledku prezentovaného programem.

- **PASS**
  Takový test zařízení splnilo.

- **FAIL**
  Zařízení nesplnilo daný test, důvod je uveden v logu.

- **INCONCLUSIVE**
  Nelze rozhodnout automaticky, uživatel musí ručně zkontrolovat správnost logu a paketů, které šly po síti. U všech zde uvedených jsem při kontrole nenarazil na problém a proto i tento výsledek považuji za splnění testu.

- **ERROR**
  Během testu došlo k chybě, v tomto případě test také neprojde. Tento výsledek se vyskytuje typicky pokud controller nenalezne zařízení.

7.2.1 Karta cifX

Verze s kartou cifX je hlediska komunikace po PROFINETu nejhorší, toto je dáno zejména použitím nejstarší verze firmware.

Pro testování byl použit předkompilovaný firmware `cifxrepns.nxm` verze 2.1.3.1 ze dne 12.6.2008 a ovladač verze 1.0 ze dne 10.11.2009 (identické s revizí 2 v SVN).

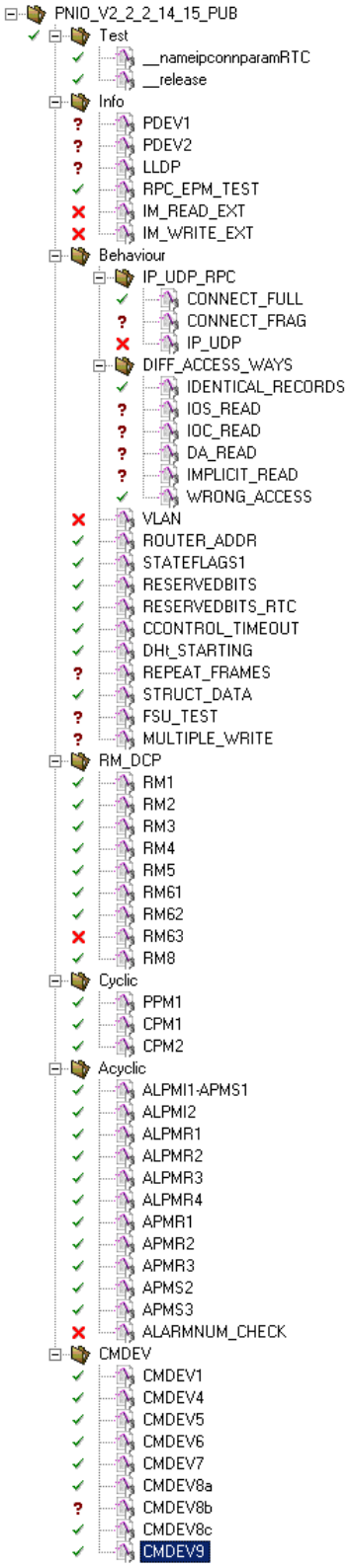
- IM_READ_EXT a IM_WRITE_EXT
  V použité verzi firmware je chyba a v paketu je poslána špatná velikost bloku I&M, proto všechny testovací případy pracující s blokem I&M neprojdou (status **FAIL**).

- IP_UDP
  Chyba ve čtení bloku I&M, stejné jako pro případ výše.

- VLAN
  Chyba kvůli bloku I&M.

- RM63
  Je závislý na případu IM_READ_EXT, který neprojde.

- ALARMNUM_CHECK
  V tomto případě je testován počet opakování alarmu v případě, že controller neodešle potvrzení. Zařízení má alarm opakovat třikrát (s jistou tolerancí), zařízení však opakuje alarm 69x (procesní) nebo 70x (diagnostický). Chyba je na straně firmware, počet opakování nelze z host CPU ovlivnit.



Obrázek 7.3: Výsledek testu pro cifX

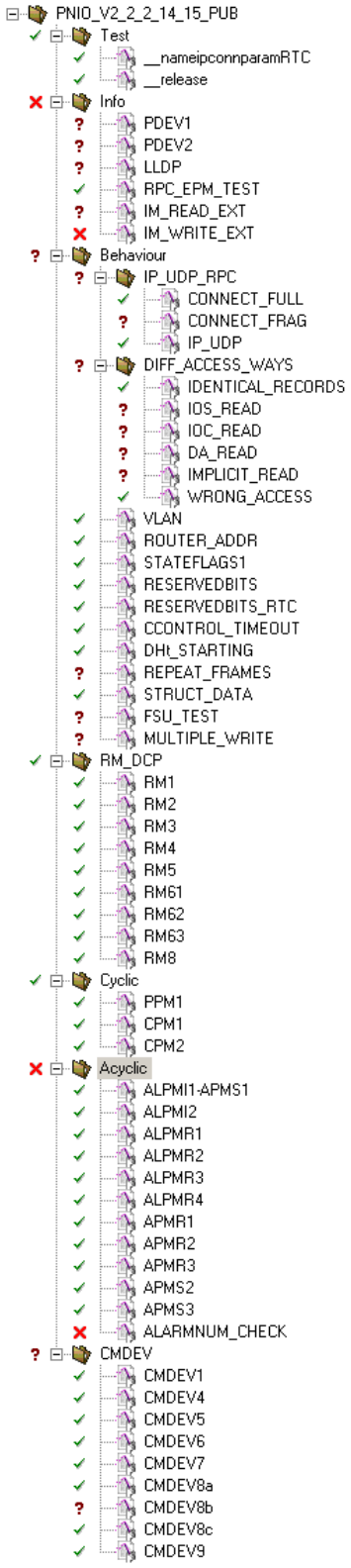
7.2.2 Modul comX, Linux

Verze ovladače pro modul comX je z hlediska komunikace po PROFINETu nejlepší. Pouze dva testovací případy selžou. Jejich popis a důvod jsou uvedeny níže.

Pro testování byl použit předkompilovaný firmware `comxrepns.nxf` verze 2.1.40.0 ze dne 9.6.2009 a ovladač verze 1.0 ze dne 10.11.2009 (identické s revizí 2 v SVN).

- IM_WRITE_EXT
  V tomto testovacím případu je testován zápis bloků I&M. V netXu je podporován pouze povinný blok I&M0, který nelze zapisovat ze strany controlleru. Toto proběhne v pořádku, ale firmware vrací nesprávný chybový kód. Tato chyba je na straně firmware a na dotaz u Hilscheru jsem dostal odpověď, že tato chyba je řešena v novější verzi frimware.

- ALARMNUM_CHECK
  V tomto případě je testován počet opakování alarmu v případě, že controller neodešle potvrzení. Zařízení má alarm opakovat třikrát (s jistou tolerancí), zařízení však opakuje alarm 60x (procesní) nebo 71x (diagnostický). Stejně jako v předchozím případě je chyba na straně firmware a je také řešeno v novější verzi.



Obrázek 7.4: Výsledek testu pro comX

7.2.3 Modul comX, PikeOS

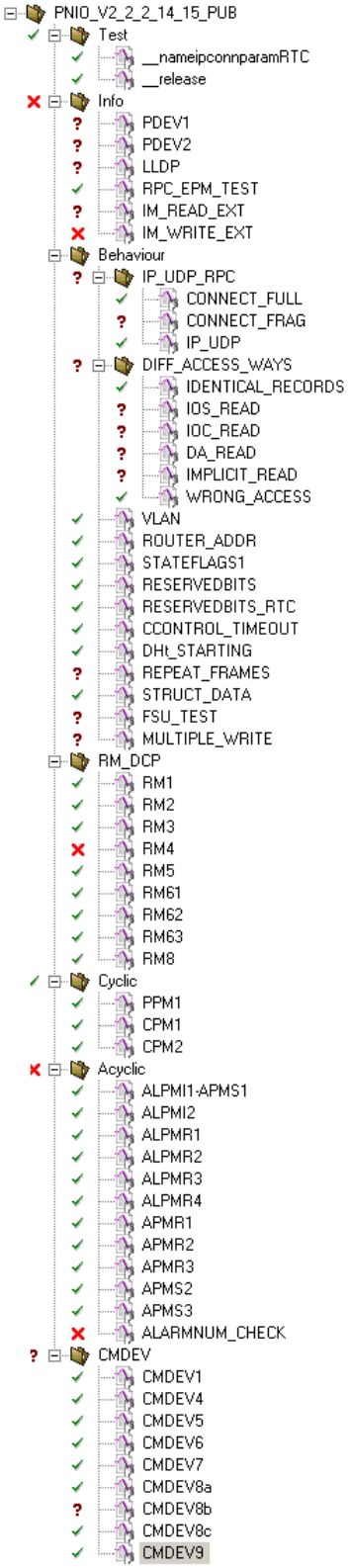
Verze ovladače pro modul comX a OS PikeOS je shodná s testování modulu comX a OS Linux. Z hlediska komunikace po PROFINETu je jen jediný rozdíl.

Pro testování byl použit předkompilovaný firmware `comxrepns.nxf` verze 2.1.40.0 ze dne 9.6.2009 a ovladač verze 1.0 ze dne 10.11.2009 (identické s revizí 2 v SVN).

- IM_WRITE_EXT
  Shodné jako ve verzi pro OS Linux (viz 7.2.2).

- RM4
  V tomto testovacím případu se testuje možnost nastavení pevných parametrů zařízení (non-volatile). Controller pošle IP konfiguraci a zařízení si ji musí uložit jako pevné nastavení, které je použito při nové inicializaci. Na Linuxu je toto uloženo do souboru `remanent.dat`, na PikeOS však není k dispozici systém souborů, proto nelze uložit parametry. Po resetu netX čeká na DHCP a controller již nastavené síťové rozhraní. Případ vrátí status **ERROR**.

- ALARMNUM_CHECK
  Shodné jako ve verzi pro OS Linux (viz 7.2.2).



Obrázek 7.5: Výsledek testu pro comX

7.3 Testování doby odezvy a zátěže procesoru

Při snaze maximálně snížit dobu odezvy zařízení je nutné velmi často vyčítat blok cyklických dat. Tímto ovšem výrazně zatížíme procesor. Pro zařízení na desce SHARK byla měřena odezva a zátěž procesoru, kterou bylo třeba pro její dosažení.

Tuto zátěž ovšem **přebírá uživatelská aplikace**, ovladač jako takový vůbec nevyčítá cyklická data. Záleží tedy pouze na nadřazené uživatelské aplikaci. Ovladač na MPC5200 zatěžuje procesor pouze zhruba 1% systémového času.

Nejprve bude popsána metoda měření a poté výsledky pro modul comX.

7.3.1 Metodika měření

Měřící zapojení bylo použito shodné jako na obrázku 7.1. V zařízení IO device (modul comX) se spustí jednoduché vlákno, které vždy vyčte vstupní data a opíše je na výstup. Poté počká předem určenou dobu. V pseudokódu lze zapsat následovně.

```
while(1){
  read_IO();
  write_IO();
  sleep();
}
```

Doba spánku byla měněna v rozmezí 0-3ms. Pro každou variantu byl vygenerován log ve wiresharku.

Ke controlleru připojíme běžný komerční modul digitálních vstupů a při měření je přepínáme, abychom dosáhli změny hodnot výstupních dat controlleru. Vlastní odezvu poté určíme jako dobu mezi změnou výstupních dat controlleru a IO device.

Použitá měřící metoda není zdaleka ideální, je zde zanesena chyba od switche, počítače na kterém běží program Wireshark atd. Avšak pro porovnání obou variant plně postačuje, protože zanesená chyba je stále stejná.

Použití vlákna s definovaným intervalem spaní bylo použito zejména proto, že v POSIXovém oddílu na PikeOS nelze spustit více aplikací.

Zátěž procesoru byla měřena na Linuxu pomocí volání `top`. Pro PikeOS se bohužel nepodařilo najít efektivní určení zátěže procesoru.

7.3.2 Modul comX, Linux

Výsledky měření jsou uvedeny v tabulce 7.1.

doba spánku [ms]	zátěž procesoru [%]	odezva [ms]
0	60	1.5
1	12	2.4
2	7	3.5
3	4	7.5

Tabulka 7.1: Výsledky měření odezvy pro Linux

7.3.3 Modul comX, PikeOS

Výsledky měření jsou uvedeny v tabulce 7.2.

doba spánku [ms]	zátěž procesoru [%]	odezva [ms]
0		1.2
1		3.1
2		3.7
3		5.6

Tabulka 7.2: Výsledky měření odezvy pro PikeOS

Kapitola 8

Závěr

Tato práce popisuje vývoj komunikační periferie pro průmyslový standard PROFINET IO RT. Pro periferii bylo použito řešení založené na specializovaném komunikačním čipu. Zvoleným čipem je čip netX německé firmy Hilscher.

V úvodu práce je velmi stručně popsána komunikace na PROFINETu (viz kapitola 2), následuje popis čipu netX a rozhraní, které poskytuje (viz kapitola 3). Dále je uveden popis ovladače a nakonec výsledky testování celého zařízení.

Ovladač byl vyvinut ve třech variantách. První je použití karty cifX na běžném PC s OS Linux (viz kapitoly 4.1 a 5). Tato varianta sloužila zejména v začátku vývoje, kdy ještě nebyla k dispozici deska SHARK. Je také uživatelsky nejpřívětivější, protože při vývoji lze používat veškeré služby a výkon PC. Druhou variantou je použití modulu comX na desce SHARK s použitím OS Linux (viz kapitoly 4.2 a 5). Tato verze byla hlavní v pozdních fázích vývoje. Poslední variantou je použití modulu comX na desce SHARK s použitím OS PikeOS. Tato varianta je pro vývoj nejméně přívětivá, protože kvůli každé i sebemenší změně je nutné překompilovat celé jádro PikeOS a restartovat desku SHARK.

Při vývoji ovladače se vyskytlo několik výrazných problémů. Nejvýraznějším je fakt, že procesor MPC5200 je architektury big endian a čip netX je architektury little endian. Oba jsou spojeny adresně datovou sběrnicí a pořadí bajtů bylo častou chybou při ladění. Dalším byla absence vyčerpávající dokumentace k modulu comX, kde zcela chyběl jakýkoli popis jak jej zprovoznit. Jedním z již méně výrazných byly změny verzí dokumentů od Hilscheru, kde v prvních verzích manuálů chyběly popisy (týkalo se zejména inicializace).

K zařízení byl napsán soubor GSDML s popisem zařízení, který je využíván PROFINET IO controllerem při konfiguraci HW.

Výsledné zařízení umožňuje host CPU komunikovat po PROFINETu bez nutnosti zpracovávat stack, čímž výrazně ulehčuje práci procesoru. Ten díky tomu může vykonávat další operace. Ovladač při použití v host CPU MPC5200 zatěžuje procesor zhruba 1% systémového času, zbytek výkonu je stále k dispozici.

Celé zařízení také splňuje drtivou většinu testovacích případů podle specifikace PI (viz kapitola 7.2). Díky tomuto lze nasadit do reálného provozu.

Dalšími úpravami v budoucnosti může být zejména přechod na novější verzi firmware pro netX, jádro Linuxu nebo verzi PikeOS.

Práce byla prezentována na výstavách Embedded world 2009 (na stánku katedry řídící techniky) a Ampér 2009 (na stánku firmy Mikroklima).

Literatura

[1] Popp M., Weber K.: *The Rapid Way to PROFINET*
PROFIBUS Nutzeorganization e.V. 2004

[2] Matiášek P.: *Diplomová práce, Komunikace Profinet IO,* 28.5.2006

[3] Mezera P.: *Diplomová práce, Vývoj zařízení Profinet IO,* 18.1.2008

[4] Hilscher GmbH
`http://www.hilscher.com` [cit. 10.3.2009]

[5] Freescale semiconductors:
`http://www.freescale.com` [cit. 22.7.2009]

[6] Mirkoklima:
`http://www.midam.cz/` [cit. 3.11.2009]

[7] Corbet J., Rubini A., Kroah-Hartman G.: *Linux Device Drivers 3.edition*
O'Reilly Media, Sebastopol, 2005
ISNB: 0-596-00590-3

[8] Freescale semiconductors: *MPC5200 Users Manual,* rev. 3, 12.03.2006
`MPC5200UM.pdf`

[9] Hilscher GmbH: *Protocol API for PROFINET IO RT Device,* rev. 13, 27.11.2008
`PROFINET IO RT Device Protocol API.pdf`

[10] Hilscher GmbH: *netX Dual-Port Memory Interface,* rev. 7, 3.12.2008
`netX_DPM_Interface_rev7.pdf`

[11] Hilscher GmbH: *netX product overview,* rev. 0.9, 19.4.2007
`netX_Product_brief.pdf`

[12] SYSGO AG: *PikeOS Personality Manual: POSIX,* verze S1729-1.5, PikeOS v2.2
`posix.pdf`

[13] SYSGO AG: *PikeOS System Software Reference Manual,* verze S1729-1.5.2.2
`psswref.pdf`

[14] PROFIBUS Nutzerorganisation e.V.: *GSDML Specification for PROFINET IO,* verze 2.20, 15.6.2008
`GSDML_Spec_2352_V220_Jul08.pdf`

[15] program Wireshark, verze 1.0.2
`http://www.wireshark.org` [cit. 20.11.2009]

[16] program PROFINET IO Tester, verze 1.5 revize 4282
`http://www.pn-tester.de/index_en.htm` [cit. 15.7.2009]

[17] : *PROFINET IO Tester Product documentation,* verze 1.5, 9.4.2009
`ProductDoc.pdf`

[18] PROFIBUS Nutzerorganisation e.V.: *Profile Guidelines Part 1: Identification & Maintenance Functions,* verze 1.16 draft, 5.11.2008
`Profile-Guidelines-I_M_3502_d116_Nov08.pdf`

[19] HW wiki na katedře řídící techniky
`http://rtime.felk.cvut.cz/hw/index.php` [cit. 18.12.2009]

[20] program Doxygen, verze 1.5.5
`www.doxygen.org` [cit. 5.12.2009]

[21] SYSGO AG: *PikeOS Tutorials,* verze S1729-1.4, PikeOS v2.2
`tutorial.pdf`

Příloha A

Přiložené soubory

A.1 Zdrojové kódy

Veškeré zdrojové kódy jsou umístěny ve složce `src`. Pro každou variantu ovladače je vlastní složka, přiložené zdrojové kódy odpovídají verzi 1.0, která je identická s revizí 2 v SVN.

Zdrojové kódy jsou dále umístěny v systému správy verzí SVN. Tato SVN však není veřejně přístupná, v případě potřeby přístupu kontaktujte pana Burgeta.

Adresa SVN je `okean.eusophos.cz/svn/profinet-io/`.

A.2 Manuály uvedené v seznamu literatury

Všechny manuály v elektronické podobě jsou přiloženy ve složce `manual`.

A.3 Dokumentace vygenerovaná pomocí Doxygenu

Pro všechny projekty byla vygenerována dokumentace pomocí Doxygenu [20]. Tato dokumentace je ve formátu `html` a slouží jako referenční příručka pro zdrojové kódy. Zde lze nejpohodlněji vyčíst podrobné informace o každé funkci jako například parametry a vzájemné propojení.

Veškerá dokumentace je umístěna na CD ve složce `doxygen/`. Zvolený formát html je pro použití nejvýhodnější z možností Doxygenu. Tímto se ovšem vytvoří velké množství souborů, proto je vždy dokumentace zabalena v `*.zip` archivu.

A.3.0.1 Kernel modul pro PC

Archiv `kernel_pc.zip`.

A.3.0.2 Kernel modul pro pPC

Archiv `kernel_ppc.zip`.

A.3.0.3 C-toolkit pro PC

Archiv `ctool_pc.zip`.

A.3.0.4 C-toolkit pro powerPC

Archiv `ctool_ppc.zip`.

A.3.0.5 DPM API

Archiv `dpm_api.zip`.

A.3.0.6 Ovladač zařízení pro PikeOS

Archiv `pike.zip`. Pro PikeOS je veškerá dokumentace v jednom, není rozdělena na jednotlivé části a obsahuje i popis DPM API, které je shodné s verzí pro Linux.

A.4 Pomocné aplikace

V rámci této práce bylo napsáno několik menších aplikací, které neslouží přímo pro komunikaci po PROFINETu, ale slouží pro podružné činnosti.

A.4.1 Aplikace pro správu firmware pro modul comX

Tato aplikace řeší správu firmware pro modul comX. Firmware je v modulu comX uložen ve flash paměti a stačí jej nahrát pouze jednou.

Pro ovladač je tedy zbytečné poskytovat funkce pro správu firmware, které pouze zabírají místo. Proto jsou v samostatné aplikaci. Možnosti této aplikace jsou následující.

- Nahrání nového firmware
  Pro zvolený kanál je nahrán nový firmware, starý je smazán.

- Smazání firmware
  Pro zvolený kanál je smazán soubor firmware z modulu comX.
- Uložení firmware z comXu do hosta
  Pro zvolený kanál je uložen soubor firmware z modulu comX do systému souborů hosta.
- Vypsání firmware uložených v comX
  Vypsání všech souborů uložených ve flash paměti modulu comX.
- Vypsání informací o firmware
  Pro zvolený kanál vypíše soubor firmware s identifikací (tj. verze, datum....).
- Porovnání firmware
  Porovná hash firmware uloženého v modulu comX s hash kódem firmware uloženého v hostu.

Aplikace je umístěna ve složce `comX_Linux/DPM_toolkit/development/fw_api/` a počítá se u ní s úpravami podle potřeb uživatele (přepsání funkce `main.c`). Předpokládané použití aplikace je její spuštění ze sdíleného systému souborů NFS z PC, nahrání nového firmware a opětovné spuštění desky SHARK z flash.

I pro tuto aplikaci je vygenerována referenční dokumentace v Doxygenu, konkrétně se jedná o archiv `fw_app.zip` ve složce `doxygen`.

A.4.2 Konfigurační aplikace pro kartu cifX

Tato aplikace byla napsána pro naplnění konfiguračních souborů v instalačním adresáři pro kartu cifX. Cílem bylo vyhnout se ručnímu vypisování souborů.

Tato aplikace však není příliš používána, parametry jsou totiž zapsány pevně v kódu, proto je nutné ji pokaždé zkompilovat. Dále změny menšího rozsahu je výhodnější psát ručně.

Hlavním důvodem pro přípravu této aplikace byla příprava souborů `warmstart.dat`, které byly využívány v prvotních verzích pro inicializaci. V současné podobě jsou i parametry pro warmstart paket plněny v aplikaci a tento soubor nemá význam.

A.5 Logy z testování zařízení

Logy z testování zařízení jsou umístěny ve složce `test`.

A.5.1 Logy pro testování s S7-300

Log je umístěn ve složce `test/S7-300`. Je uveden pouze jeden případ, v souboru `*.log` je uveden výpis konzole host CPU a v souboru `*.pcap` je uveden sled paketů z programu wireshark. Pro všechny varianty ovladače je log shodný.

A.5.2 Logy pro testování s PROFINET IO Testerem

Log je umístěn ve složce `test/PN IO tester`. V jednotlivých složkách jsou umístěny kompletní logy podle kapitoly 7.2.

A.5.3 Logy pro testování odezvy

Pro zařízení byl vygenerován log (pouze záznam paketů z Wiresharku) z měření odezvy v závislosti na zátěži procesoru. Ten je umístěn ve složce `test/latency`.

A.6 Soubor GSDML

Soubory GSDML pro kartu cifX i modul comX jsou přiloženy a to ve složce `GSDML`.

A.7 Plakát pro výstavu Embedded world

Práce byla vystavena na mezinárodní výstavě Embedded world 2009 v Norimberku v rámci stánku katedry řídící techniky. A to formou posteru, který je přiložen v příloze (soubor `ew_2009_poster_hw.pdf`).