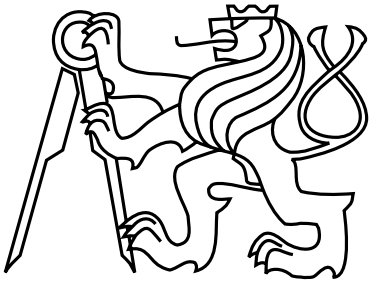
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

VoIP na FPGA

Martin Tůma

Vedoucí práce: Ing. Martin Daněk, Ph.D.

Studijní program: Elektrotechnika a informatika magisterský

Obor: Informatika a výpočetní technika

květen 2008

Poděkování

Děkuji vedoucímu diplomové práce Ing. Martinu Daňkovi, Ph.D. za cenné rady, připomínky a metodické vedení práce. Dále bych chtěl poděkovat Ústavu teorie informace a automatizace AV ČR, v.v.i. za možnost práce na přípravku ML403 a katedře počítačů FEL ČVUT za dlouhodobé vypůjčení vývojové desky Spartan-3E starter kit board.

Prohlášení

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

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

V Praze dne 15.5.2008 .....

Abstract

The work presents a design of a device for point-to-point, full duplex, audio data transmission over the Ethernet, implemented on the ML403 FPGA development board. The implementation is based on a system-on-chip (SoC) system with the Xilinx MicroBlaze microprocessor, and it is compatible with the SIP VoIP protocol.

Abstrakt

Práce prezentuje návrh zařízení realizující dvoubodový, plně duplexní přenos audio dat po počítačové síti Ethernet implementované na FPGA přípravku ML403. Implementace je založena na systému na chipu (SoC) s mikroprocesorem MicroBlaze a je kompatibilní s VoIP protokolem SIP.

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

AC97	Audio Codec '97
ADC	Analog-to-Digital Converter
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN	Abstract Syntax Notation
BNF	Backus-Naur Form
BRAM	BlockRAM
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
DAC	Digital-to-Analog Converter
DDR	Double Data Rate
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DP	Diplomová Práce
EDK	Embedded Development Kit
ELF	Executable and Linkable Format
ENUM	tElephone NUmber Mapping
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
GCC	GNU Compiler Collection
GNU	GNU's Not Unix
HTTP	Hypertext Transfer Protocol
HW	Hardware
I/O	Input/Output
IETF	Internet Engineering Task Force
IP	Internet Protocol
IP core	Intellectual Property core
IPC	Inter-Process Communication
IRQ	Interrupt
ISC	Internet Systems Consorcium
ISDN	Integrated Services Digital Network
ITU	International Telecommunication Union
LAN	Local Area Network
LCD	Liquid Crystal Display
MAC	Medium Access Control(ler)
MMU	Memory Management Unit
MPU	MicroProcessor Unit
OPB	On-chip Peripheral Bus
OS	Operační Systém
PCM	Pulse-Code Modulation
PCMA	ITU G.711 PCM A-Law Audio 64 kbit/s
PCMU	ITU G.711 PCM μ-Law Audio 64 kbit/s
PER	Packed Encoding Rules
PHY	PHYsical layer (controller)

POSIX Portable Operating System Interface
PSTN Public Switched Telephone Network
RAM Random Access Memory
RFC Request For Comments
RISC Reduced Instruction Set Computer
RTCP RTP Control Protocol
RTP Real-time Transport Protocol
SDK Software Development Kit
SDP Session Description Protocol
SIP Session Initiation Protocol
SW Software
UDP User Datagram Protocol
URI Uniform Resource Identifier
VHSIC Very-High-Speed Integrated Circuits
VHDL VHSIC Hardware Description Language
VoIP Voice over IP
WWW World Wide Web
XMPP eXtensible Messaging and Presence Protocol

1 Úvod

VoIP – Voice over Internet Protocol – tedy přenos zvuku pomocí Internet protokolu, nebo obecněji přenos zvuku přes datové sítě, se stává v současné době díky rychlému rozvoji těchto sítí alternativou k volání přes „klasické“ telefonní sítě (PSTN). VoIP umožňuje sjednocení infrastruktury pro přenos hlasu a dat a tím tuto infrastrukturu zjednodušit a snížit tak náklady na její vybudování a údržbu. Díky dnes běžně praktikovanému paušálnímu účtování za datové služby přináší VoIP koncovým zákazníkům navíc možnosti výrazných úspor na telefonních hovorech, které mohou být v případě mezinárodních hovorů i v řádu stovek procent.

Oblasti vhodné pro nasazení VoIP sahají od vnitropodnikové telefonní sítě nadnárodní korporace až po osobní počítače domácích uživatelů. Prudký nárůst kvantitativních (přenosová rychlost) i kvalitativních (zpoždění, ztrátovost paketů) parametrů datových sítí včetně Internetu v mnoha případech již eliminoval problémy vyplývající z podstaty VoIP hovorů – přenosu hlasu skrze datagramové služby v přepínaných sítích. V dostatečně dimenzovné propojovací síťi postavené na některé ze současných technologií by se neměly vyskytovat ztráty paketů či velká zpoždění způsobená přetížením sítě či technologickými vlastnostmi sítě. Taktéž dostupná přenosová rychlost je u většiny dnes používaných připojovacích metod[1] pro VoIP dostatečná. Trend vývoje pak směřuje právě k rychlejším a spolehlivějším datovým sítím/Internetu, a tak je na IP telefonii mnohými nahlíženo jako na telefonní službu budoucnosti.

Koncová zařízení pro IP telefonii mohou být a často také jsou klasické počítače, vybavené patřičným programem, v terminologii VoIP nazývaným softphone, doplněné o headset (sluchátko s mikrofonem). Z praktických důvodů (spotřeba, rozměry, jednoduché intuitivní ovládání, ...) se však postupně začínají prosazovat speciální zařízení odpovídající běžnému telefonnímu aparátu. Tato zařízení však nejsou interně nic jiného, než jednoduché mikroprocesorové systémy, tedy opět počítače, při jejichž návrhu je snahou do jednoho integrovaného obvodu „vměstnat“ co nejvíce logiky. Realizovat na jediném chipu jak mikroprocesor, tak skrze integrovanou sběrnici připojené řadiče periferií/periferie samotné, umožňuje dosáhnout vysoké míry integrace zařízení a s tím spojených výhod (cena, spotřeba, spolehlivost). O zařízeních tohoto typu se pak obecně mluví jako o SoC (System on Chip) systémech.

Programovatelná hradlová pole (FPGA) umožňují jednoduchý návrh a testování takovýchto systémů. Výrobci FPGA jednak produkují obvody s již integrovanými CPU jádry (PowerPC v obvodech firmy Xilinx, ARM v obvodech firmy Altera, ...), a dále jsou, ať už přímo od výrobců FPGA či „třetích stran“, dostupné různé IP cores implementující vhodný RISC procesor, který je možné jednoduše propojit se zbytkem logiky implementované v FPGA. Příkladem takového IP core budiž třeba v zadání DP uvedený CPU MicroBlaze.

1.1 Cíle práce

Cílem této diplomové práce je návrh a realizace koncového VoIP zařízení – VoIP klienta – na FPGA ve formě SoC systému tak, aby splňoval parametry dané zadáním DP. V širších souvislostech to znamená provést tyto základní úkony:

- Analýzu existujících VoIP protokolů, zařízení a principů jejich fungování.
- Analýzu možností dané SW a HW platformy a příslušných vývojových prostředků.
- Návrh struktury zařízení, rozdělení jednotlivých funkcí mezi SW a HW.
- Realizaci/implementaci zařízení a jeho otestování vzhledem k požadovaným vlastnostem/parametrům.

[1] Výjimkou jsou v obou případech mobilní sítě – WiFi, GPRS, CMDA, ..., kde jsou možnosti provozu VoIP stále omezené.

Hlavní váha práce spočívá v realizační části, cílem práce není provést hlubokou analýzu IP telefonie jako takové. Principy a z nich vyplývající vlastnosti přenosu zvuku ve směrovaných sítích jsou tak rozebírány pouze v kontextu návrhu zařízení.

1.2 Poznámky k implementaci a textu DP

Jako vývojová deska/přípravek, na kterém je zařízení realizováno, byla díky nejlepší dostupnosti pro autora zvolena deska Xilinx ML403. Tento konkrétní typ sice není uveden v zadání jako možná platforma pro realizaci, jedná se ale o desku prakticky identickou s modelem ML401. Oba přípravky se liší pouze v typu použitého Virtex-4 chipu a i výrobce poskytuje pro obě desky společnou dokumentaci. Typ použitého chipu je pro realizované zařízení irelevantní, neboť toto nevyužívá žádnou z vlastností, ve které se FPGA na obou deskách liší.

Systém je navíc navržen tak, aby při případném doplnění o rozšiřující modul s AC97 kodekem fungoval i na deskách s FPGA Spartan-3 – Spartan-3E starter kit board.

Veškeré práce s Xilinx EDK/ISE v rámci DP probíhaly s verzí 9.1, nejnovější verzí EDK dostupné na KP FEL ČVUT v době zahájení prací na DP. V textu uvedená funkcionalita je tak garantována pouze s verzí 9.1 ISE/EDK i když je pravděpodobné, že systém lze úspěšně sestavit i s novější verzí EDK/ISE. Dále je možné, že v nových verzích vývojového prostředí jsou k dispozici některé komponenty, které jsou v rámci DP realizovány vlastními prostředky. Mohla by se naskýtat otázka, proč nejsou použity standardní součásti EDK/ISE. V době realizace však tyto alternativy neexistovaly.

V textu DP jsou velmi často používány anglické výrazy a to nejenom pro termíny, které v českém jazyce nemají ekvivalent, ale i pro termíny, jejichž (odborný) překlad existuje, ale je velmi málo rozšířen. „Multimediální datový proud“ je například označován výrazem „stream“, „vestavěný systém“ výrazem „embedded systém“ atp. Cílem je snaha eliminovat případné nejednoznačnosti vzniklé „umělým“ překladem termínů.

Anglický jazyk je dále použit u veškerých schémat a obrázků a to jak vlastních, tak přejatých. Důvod tohoto „opatření“ je ten, že některá schémata vzniklá jako součást DP jsou (nebo mohou být) využita i v anglicky psané dokumentci příslušných SW/HW modulů. Přejaté materiály jsou pak v drtivé většině také z anglicky psaných zdrojů. Úroveň jazyka potřebná k porozumění schématům, obrázkům či jejich legendám by však neměla přesáhnout elementární úroveň znalostí anglického jazyka nutnou k dosažení příslušné odborné úrovně nutné pro porozumění vlastnímu textu.

Text nemá zcela striktně definované typografické konvence, přesto lze uvést některá základní pravidla, která jsou v textu dodržována. Veškerá jména funkcí, knihoven, souborů či portů, jsou psána pomocí `neproporcionálního písma`. Speciálním případem jsou názvy implementovaných SW knihoven. Ty jsou v textu vždy nazývány jmény hlavičkových souborů knihovny čímž jsou odlišeny od systémových knihoven u nichž, nevyžaduje-li situace uvést hlavičkový soubor, je udáván pouze název knihovny. Je-li v textu zmíněn nějaký SW/HW produkt/projekt, je v poznámce na spodním okraji stránky obvykle uveden odkaz na WWW stránky projektu/produktu. Citace a odkazy jsou uváděny způsobem běžným pro odbornou literaturu.

2 Analýza a návrh řešení

2.1 Analýza současných VoIP protokolů

Tato sekce uvádí krátký souhrn v současné době používaných VoIP protokolů, jejich vlastností a rozšíření (podíl na trhu). Přestože zadání kompatibilitu s některým z existujících VoIP protokolů nepožaduje, je snahou takto kompatibilní zařízení navrhnout, neboť tím podstatně vzrostou možnosti reálného využití navrženého zařízení.

2.1.1 H.323

H.323 je standard (doporučení) definovaný Mezinárodní telekomunikační unií (ITU) jehož první verze byla uveřejněna roku 1996. H.323 není samostatně využitelný protokol ale jedná se o „zastřešující“ standard, pod který spadá celá řada samostatných protokolů. Jejich strukturu, označovanou jako H.323 stack, znázorňuje obrázek 2.1. Minimální implementace H.323 stacku umožňující realizovat VoIP hovor musí obsahovat tyto protokoly:

H.225 – hovorová signalizace.

H.245 – protokol pro vyjednání parametrů multimediálních kanálů.

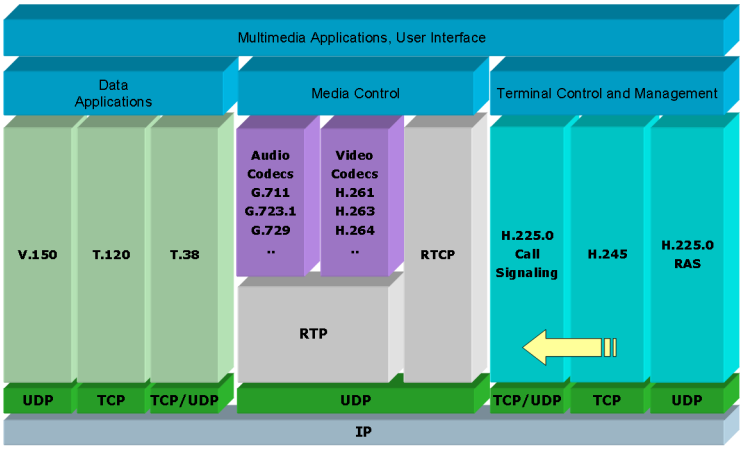
RTP – protokol pro přenos multimediálních dat v reálném čase.

Media codec – protokol definující formát/kompresi přenášeného zvuku (např. G.711).

Běžně se v rámci H.323 dále používají další rozšíření jako H.235 – bezpečnostní a ověřovací mechanismy či H.450 – doplňkové služby (call transfer, call hold, . . .). H.323 není omezen pouze na přenos zvuku, ale v závislosti na použitých kodecích umožňuje téže videohovory/videokonference či přenos textových dat. Pro přenos zvuku se nejčastěji používá některý z kodeků G.711, G.729 či G.726, pro přenos videa pak H.261, H.263 či H.264.

Protokoly ITU (H.XXX) používají definiční jazyk ASN.1 a zprávy se pro přenos kódují metodou ASN.1 PER. Protokol RTP, který nepochází od ITU, ale IETF, používá vlastní způsob binárního kódování.

H.323 definuje následující druhy zařízení:



Obrázek 2.1: Struktura H.323 stacku. Zdroj: [10].

Terminál – vlastní HW nebo SW telefon (videokonferenční systém).

Brána – zařízení, které umožňuje obousměrnou komunikaci se zařízeními v jiné komunikační síti (ISDN, analogová telefonní síť, jiná H.323 síť).

Konferenční jednotka – zařízení obsluhující hovorovou signalizaci a multimediální kanály pro konferenční spojení.

Gatekeeper – zařízení, která poskytuje služby překladu adres a řízení provozu v H.323 síti.

Pro terminály, brány a konferenční jednotky se souhrnně používá termín koncové zařízení (endpoint). Pokud je v síti gatekeeper, je centrálním řídícím prvkem H.323 sítě. Všechna koncová zařízení se pak na gatekeeperu registrují, řídí se jeho pokyny (např. povel k ukončení hovoru) a veškeré akce jsou nejdříve „konzultovány“ s ním (např. před pokusem o spojení je koncovým zařízením na gatekeeper zaslána žádost o povolení hovoru).

Standard umožňuje pro adresaci zařízení použít celou řadu mechanismů: E.164 (číslovací plán používaný pro mezinárodní telefonní síť), vlastní H.323 URI, ISUP (ISDN User Part) či dokonce emailovou adresu.

Nejjednodušší schéma H.323 spojení je přímé spojení mezi dvěma koncovými zařízeními (terminály) a probíhá takto:

1. Navázání hovoru protokolem H.225.0. Signalizace začíná zprávou `Setup` (požadavek na hovor). Hovor je přijat zprávou `Connect`, nebo odmítnut zprávou `ReleaseComplete`.
2. Dohodnutí parametrů pro přenos zvuku/videa (kodeky, IP adresy, UDP porty) protokolem H.245.
3. Zahájení přenosu zvuku/videa protokolem RTP.

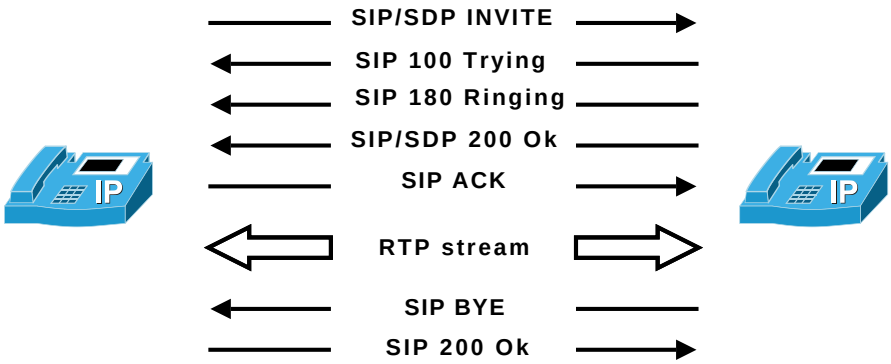
Standard H.323 je velice komplexní systém stavějící na mnoha samostatných, telekomunikačních protokolech. Je nasazován zejména v sítích velkých telekomunikačních operátorů, čemuž odpovídá i jeho podíl na celkovém objemu celosvětové VoIP komunikace, který v roce 2006 činil 80% [9]. Jeho rozšíření v koncových zařízeních – VoIP telefonech – je pro jeho složitoost nicméně podstatně menší než u konkurenčního SIPu [2]. Z tohoto důvodu také, minimálně v ČR, většina poskytovatelů VoIP (UPC, Volný, 802.cz, ...) jako komunikační protokol pro koncová zařízení volí protokol SIP.

2.1.2 SIP

Protokol SIP (Session Initiation Protocol) na rozdíl od H.323 nepochází z oblasti telekomunikací, ale je spravován „počítačovou“ organizací IETF. Přístup ke komunikaci mezi jednotlivými zařízeními je proto zcela odlišný, než u protokolu H.323. Protokol je typu klient-server (metoda výměny zpráv request-response) a svou strukturou je velmi podobný protokolu HTTP, ze kterého vychází. Zprávy jsou textové, za úvodní řádkou dotazu/odpovědi (start-line) následují hlavičky zakončené prázdnou řádkou a případné tělo zprávy.

Adresace je pak obdobná jako u ostatních internetových protokolů IETF – SIP adresa (URI) má tvar: `sip:user@host`. Možnost propojení s „klasickou“ telefonní sítí (PSTN) zajišťuje specifikace ENUM (RFC 3761), která definuje formát uživatelského jména adresy tak, aby byl převeditelný na telefonní číslo sítě PSTN.

Ani v případě SIPu k sestavení hovoru nestačí protokol samotný, ale je třeba ještě protokol SDP pro vyjednání parametrů multimediálních kanálů. Data SDP jsou však přímo součástí SIP paketu jako jeho tělo. Pro vlastní přenos zvuku/videa se pak, stejně jako u H.323, používá protokol RTP. Průběh základního, peer-to-peer, SIP VoIP hovoru udává obrázek 2.2.



Obrázek 2.2: Průběh elementárního SIP VoIP hovoru.

Kromě vlastních klientů (VoIP telefonů) se v SIP sítích mohou vyskytovat ještě další typy zařízení:

SIP server – SIP server je obvykle odpovědný za uživatele v doméně. Pracuje v proxy nebo redirect módu. V redirect módu sdělí informace o poloze volaného získané lokalizační službou zpět volajícímu a ten navazuje spojení přímo na novou adresu. V proxy módu pokračuje navazování spojení prostřednictvím serveru.

Location service – lokalizační služba je služba poskytující informace o současné poloze uživatele, tedy o jeho aktuální IP adrese.

Registrar – registrační server zpracovává požadavky o registraci od klientů a přijaté informace o poloze klienta předává lokalizační službě, která obsluhuje příslušnou oblast.

Všechna tato zařízení mohou být, a obvykle také jsou, sloučena dohromady v jeden server/program. Rozdělení funkcí je čistě logická záležitost.

Jako kodek pro přenos zvuku je nejčastěji používán G.711 (64 kb/s), na pomalejších linkách pak G.726 (16-40 kb/s) či dokonce GSM (4,75-12,2 kb/s).

Protokol SIP je díky své, oproti H.323, relativní jednoduchosti oblíben jako protokol pro „běžná“ koncová zařízení a je podporován nejširším spektrem SW a HW VoIP telefonů i VoIP operátorů.

2.1.3 Ostatní protokoly

Skype

Skype je program pro VoIP komunikaci vyvíjený firmou Skype Technologies S.A. s vlastním komunikačním protokolem. Skype se stal díky jednoduchosti použití a bezproblémovému fungování na počítačích bez veřejné IP adresy pro mnoho uživatelů synonymem pro VoIP. Protokol je ale uzavřený (jeho specifikace není veřejně přístupná), a proto pro jakoukoliv implementaci zcela nevhodný.

Google Talk (Jingle)

Jingle je rozšíření instant messaging protokolu Jabber/XMPP umožňující p2p multimediální přenosy v reálném čase. Protokol byl vyvinut firmou Google pro službu Google talk a je nyní ve fázi standardizace organizací XMPP Standards Foundation. V současné době podporují Jingle kromě Google talk i někteří další Jabber klienti (Kopete, Jabin), většinou však pouze „experimentálně“.

2.2 Analýza možných SW/HW platforem

Struktura systému na vývojové desce ML403 může mít celou řadu podob. Od čistě FPGA řešení, kde je veškerá funkcionalita implementována v HW, až po „plnohodnotný“ operační systém (linux) využívající v FPGA (Virtex-4) integrované CPU PowerPC, kde je vlastní funkcionalita zařízení čistě SW záležitostí. Možnosti a výhody/nevýhody jednotlivých platforem shrnuje následující sekce. Jelikož HW platforma je víceméně daná zadáním (systém s CPU MicroBlaze), jsou zde rozebrány hlavně možnosti SW platforem.

2.2.1 Hardware

Virtex-4 spolu s příslušnými Xilinx ISE/EDK nástroji umožňuje sestavit 3 typy hardwarových platforem:

1. FPGA – systém bez CPU, využívající FPGA pouze jako logický obvod.
2. MicroBlaze – MicroBlaze je CPU sestavený z logických bloků FPGA.
3. PowerPC – na FPGA integrovaný 32-bitový CPU.

Systém bez CPU logicky nemá žádnou podporu pro „klasický“ programovací jazyk (C, C++, ...), podporován je pouze návrh systému ve VHDL/Verilogu. Takovýto systém sice teoreticky umožnuje navrhnout nejvýkonnější, pro daný problém zcela optimalizované řešení, kvůli složitému vývoji je však vhodný pouze v případech, kdy potřebujeme maximálně výkonné řešení s minimálními HW nároky.

Na opačné straně je systém s integrovaným CPU PowerPC 405. PowerPC firmy IBM je „klasický“ 32-bitový RISC procesor, který je podporován širokým spektrem překladačů i operačních systémů. Na takový systém je i díky dostatečné velikosti paměti DDR RAM obsažené na desce možné nahlížet jako na plnohodnotný počítač a provozovat na něm celou řadu embedded operačních systémů. Ve vlastní logice FPGA je pak možné implementovat nejrůznější sběrnice, řadiče periferií či periferie samotné.

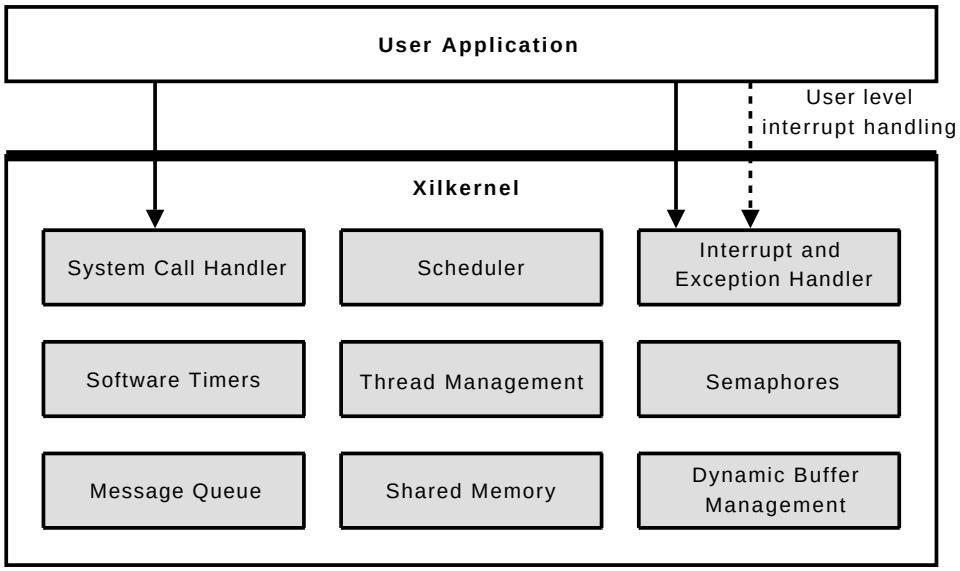
Na pomezí mezi platformami uvedenými výše je pak systém s CPU MicroBlaze. MicroBlaze je 32-biový RISC procesor, který je možné sestavit z běžných logických bloků FPGA. Umožňuje tak návrh procesorových systémů nejenom na Xilinx FPGA řady Virtex, ale i na levných FPGA Spartan-3. Nevýhodou oproti systému s CPU Virtex-4 je znatelné snížení dostupných logických bloků pro periferie na chipu a nižší výkon CPU.

MicroBlaze má 3 nebo 5 stupňovou pipeline[1], aritmeticko-logické operace provádí v jednom taktu, load/store operace ve dvou a skoky maximálně ve třech taktech. Maximální rychlost CPU na Virtex-4 FPGA je dle údajů výrobce [7] 160 MHz (100 MHz na Spartan-3 FPGA) a výkon 184 DMIPS (115 DMIPS) v Dhrystone 2.1 benchmarku. Některé části procesoru jsou konfigurovatelné/volitelné (datová a instrukční cache, floating-point jednotka, HW násobička, ...), konkrétní zvolená konfigurace je rozvedena v kapitole 3.

2.2.2 Standalone system

Anglickým termínem *standalone system* je nazývána softwarová platforma bez jakéhokoliv operačního systému, tedy platforma implementující pouze nejzákladnější programovou podporu. V případě MicroBlaze a pro něj portovaného překladače GCC – mb-gcc to znamená podporu standardní C knihovny `libc`, podporu práce s pamětí a funkce specifické pro procesor (malloc, funkce pro práci s HW přerušeními a HW výjimkami) v knihovně `libxil` a základní I/O funkce (knihovna `stdio`).

[1] Záleží na konfiguraci CPU, zda je optimalizován na rychlost či na plochu.



Obrázek 2.3: Struktura OS Xilkernel.

Absence jakéhokoliv OS znamená nulovou podporu pro vícevláknové aplikace, na druhou stranu umožňuje nejlepší kontrolu nad průběhem (časovými omezeními) programu, což bývá u embedded aplikací často zásadním požadavkem.

Pro VoIP aplikaci je z principu věci nutná podpora protokolu IP. Pro standalone systém je v Xilinx EDK k dispozici knihovna `lwip`, respektivě její RAW API. To sestává z funkcí pro nízkoúrovňovou manipulaci s TCP/IP stackem, které je nutné v programu periodicky volat.

2.2.3 Xilkernel

Kromě nástrojů a knihoven uvedených výše, obsahuje Xilinx EDK i jádro/operační systém Xilkernel. Xilkernel je vysoce modulární, real-time kernel s POSIX API pro systémy s CPU PowerPC a MicroBlaze. Základní funkce poskytované kernelem jsou:

- POSIX vlákna s round-robin a prioritním plánováním (scheduling).
- POSIX synchronizační služby – semafory a mutexy.
- POSIX IPC služby – fronty zpráv a sdílená paměť.
- Kernelem spravovaná alokace paměti z dynamického buffer poolu.
- Softwarové časovače.
- API pro uživatelské zpracování přerušení.

Je zřejmé, že hlavním účelem Xilkernelu je podpora vícevláknových aplikací. Xilkernel umožňuje rozdělit aplikaci na samostatné logické celky (vlákna) řešící jednotlivé části problému a ty pak implementovat samostatně. Odpadá tak nutnost vlastními prostředky řešit přepínání mezi více funkčními bloky aplikace typické pro kód používaný v mikrokontrolérech/jednočipech.

Naopak správa HW prostředků, po správě procesů/vláken druhá stěžejní část operačních systémů, je podporována pouze rozhraním pro registraci rutin obsluhujících HW přerušení procesoru. Kromě ovladačů systémových prostředků nutných pro běh kernelu samotného (systémový časovač, řadič přerušení) tedy Xilkernel neobsahuje žádné ovladače periferií ani API pro jejich začlenění do kernelu. Ovladače periferií se tak programují v uživatelském prostoru stejně jako v případě standalone systému – kernel nijak neomezuje přímý přístup do paměti, kde jsou jednotlivé periferie namapovány.

Kromě výše zmíněného RAW API knihovny lwip je v systému s Xilkernelem možné využít i druhého API knihovny – sockets API, které je ekvivalentem klasického BSD sockets API (většina systémových volání je identická). TCP/IP stack pak běží ve vlastním systémovém vlákně a uživatel komunikuje se síťovým stackem pomocí klasického open-read-write-close modelu.

2.2.4 PetaLinux (MicroBlaze uClinux)

uClinux, je speciálně upravená verze linuxového jádra pro systémy bez HW podpory správy paměti (MMU). uClinux byl na University of Queensland naportován i pro CPU MicroBlaze a vznikl tak projekt MicroBlaze uClinux[2] na němž pak staví projekt PetaLinux[3].

Výsledkem je „plnohodnotný“ linux se všemi jeho možnostmi a vlastnostmi. Pro aplikace je tak k dispozici například kompletní POSIX rozhraní či BSD sockety. Navíc lze využít prakticky veškerý SW pro platformu linux, z oblasti VoIP by tak teoreticky mělo být možné použít například SIP stack PJSIP[4] (obsahuje dokonce „hotového“ SIP klienta `pjsua` se základními VoIP funkcemi) či H.323 stack OpenH323[5].

2.3 Návrh struktury systému

2.3.1 Zhodnocení analýzy

Analýzu z předchozí části můžeme shrnout do těchto stěžejních bodů:

- Použití PetaLinuxu teoreticky umožňuje sestrojit „plnohodnotné“ VoIP zařízení pouhým „poskládáním“ již existujícího SW.
- Pokud bude implementace komunikačního protokolu zařízení součástí realizace, může mít protokol v zásadě dvě podoby:

  1. Vlastní, s žádným jiným zařízením kompatibilní protokol navržený s důrazem na jednoduchou implementaci.
  2. Protokol kompatibilní s protokolem SIP (jeho minimální podmnožinu).

  jakýkoliv jiný VoIP protokol je buď extrémně složitý (H.323), experimentální (XMMP), nebo pro implementaci zcela nevhodný (Skype).
- Xilinx EDK poskytuje dvě softwarové platformy vhodné pro implementaci – standalone systém a Xilkernel. Xilkernel oproti standalone systému především zjednodušuje návrh aplikací s několika souběžně pracujícími logickými celky (vlákny).

Embedded systém postavený na linuxu je jistě efektivní a jednoduché řešení problému, které je v současné době v praxi využíváno při realizaci mnoha podobných zařízení. Toto řešení však minimálně „obchází“ ideu zadání diplomové práce – převádí problém návrhu kombinovaného HW/SW embedded systému do oblasti softwarového inženýrství, kde jsou návrhové nástroje Xilinx ISE/EDK využity pouze pro jednorázové sestavení již „hotové“ platformy. Toto řešení je také možné považovat za nejvíce náročné na HW prostředky. Linux s předpokládanými SW komponentami bude mít zcela jistě větší paměťové nároky než vysoce modulární Xilkernel (či dokonce standalone systém) a jednoduchá VoIP aplikace. Ideálu SoC, tedy systému v jednom chipu včetně veškerých pamětí, se zejména na low-level chipech Spartan-3 (viz dále), zřejmě nepodaří dosáhnout ani s úspornějšími platformami (standalone systém, Xilkernel) a bude tak

[2] http://www.itee.uq.edu.au/~jwilliams/mblaze-uclinux/
[3] http://www.petalogix.com/
[4] http://www.pjsip.org/
[5] http://www.openh323.org

nutné použít externí paměti dostupné na vývojových deskách, PetaLinux se však tomuto ideálu z uvedených možností blíží nejméně.

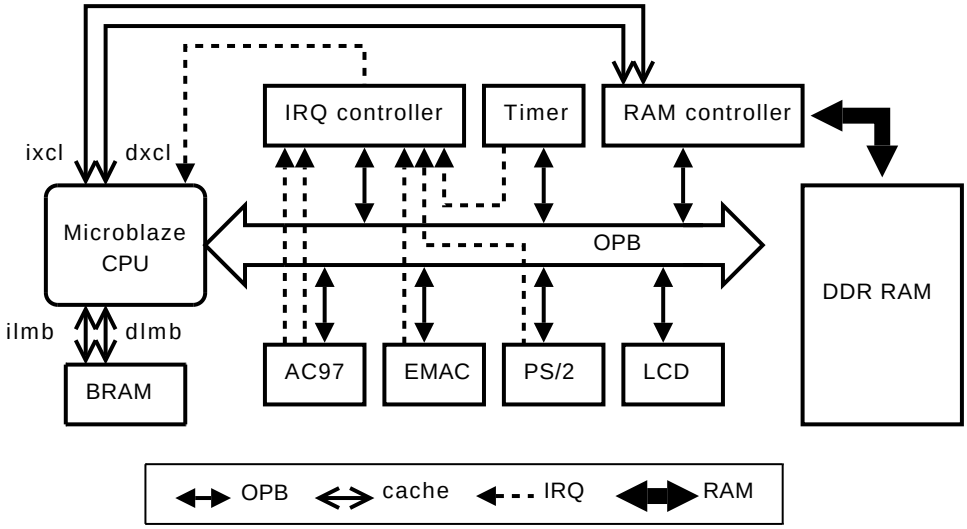
Z výše uvedených důvodů bylo rozhodnuto PetaLinuxu jako vývojové platformy nevyužít a při návrhu systému se více než na co nejširší funkcionalitu soustředit na HW náročnost systému.

V úvahu tak připadají obě dvě platformy podporované nástroji Xilinx ISE/EDK – standalone systém a systém postavený na modulárním mikrojádru Xilkernel. Vzhledem k tomu, že řešený problém ze své podstaty vede na implementaci několika (pseudo)paralelně běžících modulů (uživatelská konzole, automat stavu spojení (hovoru), automat multimediálního streamu), je vhodné použít prostředků OS pro práci s vlákny a systém postavit na Xilkernelu. Základní konfigurace Xilkernelu, implementující prakticky pouze multithreading, má dle dokumentace [4] velikost programového kódu (footprint) 7 kB, konfigurace s podporou modulů potřebných pro socket AP knihovny lwip pak 16 kB.

Velikost přibližně 16 kB má také knihovna lwip samotná. Velikost dostupné BRAM na FPGA Spartan-3E 500 (Spartan-3E starter kit board) je 32 kB, na chipu Virtex-4 (ML403) pak 64 kB. Do této paměti se za předpokladu, že bychom nechtěli či nemohli použít žádnou externí paměť, musí kromě samotného programového kódu vejít i stack a heap systému. Takovou VoIP aplikaci je pravděpodobně možné realizovat, ale zcela jistě na úkor její komplexnosti a složitosti jejího vývoje. Prakticky všechny vývojové desky s FPGA, na kterých je možné sestavit CPU MicroBlaze, disponují externí pamětí (EDK pak příslušným řadičem paměti), kterou lze pro aplikace využít. Při návrhu řešení zadání diplomové práce je proto s externí pamětí počítáno. Velikost knihovny lwip a Xilkernelu proto není rozhodující, respektive rozhodně jako negativum nepřevažuje nad pozitivy získanými jejich využitím. Toto však rozhodně neznamená, že by paměťové nároky systému byly při návrhu zcela opomíjeny! Zejména u částí, které budou mít potenciál být využívány i mimo tento projekt (především ovladače periferií), bude brán na paměťovou náročnost kódu zřetel.

Posledním „zásadním“ rozhodnutím, které je třeba učinit, je volba komunikačního protokolu VoIP zařízení. Ze dvojice {vlastní protokol ; SIP} je zvolen protokol SIP, respektive jeho podmnožina (plná implementace standardu jednoznačně přesahuje rámec i ideu zadání). Použití standardního protokolu nejenom zvýší užitnou hodnotu zařízení a tím i celé diplomové práce, ale navíc svým způsobem i usnadní vývoj a testování zařízení. Jako „protistranu“ je v tomto případě totiž možné zvolit jakýkoliv běžný SIP SW telefon (Softphone).

Jak již zmíněno, implementovaná podmnožina protokolu SIP, nebo přesněji všech tří protokolů nutných pro SIP VoIP spojení – SIP, SDP a RTP, bude zvolena tak, aby splňovala podmínky zadání DP, tedy dvoubodový plně duplexní přenos, nikoliv kompletní specifikaci protokolu(ů). V implementaci protokolu SIP tak nebude implementována žádná podpora pro registraci klienta na registračním serveru, u protokolu RTP nebude implementován jeho „sesterský“ protokol RTCP sloužící k indikaci aktuálních parametrů (kvality) multimediálního streamu atd. Podrobnosti o omezeních jsou vždy uvedeny v příslušné sekci kapitoly 3. Jako kodek pro přenášený zvuk bude pro svojí implementační i výpočetní „jednoduchost“ a své rozšíření (jeden z dvojice G.711 kodeků je podporován prakticky každým SIP klientem) zvolen G.711 A-Law a/nebo μ-law. Z reálně používaných kodeků poskytuje tento za cenu nejvyšší přenosové rychlosti (bitrate) 64 kb/s (80 kb/s na IP vrstvě) nejvyšší kvalitu zvuku.



Obrázek 2.4: Schéma HW platformy zařízení.

2.3.2 Struktura systému

2.3.2.1 Hardware

HW platforma zařízení má strukturu typickou pro systémy s CPU MicroBlaze – periferie (řadiče periferií) jsou k CPU připojeny přes sběrnici OPB. Kromě I/O periferií[6]:

AC97 – řadič zvukového kodeku AC97. Ovládá na desce obsažený zvukový kodek LM4550.

PS2 – řadič PS/2. Ovládá PS/2 port a skrze tento připojenou PS/2 klávesnici.

LCD – řadič LCD displeje. Ovládá na desce obsažený 2x16 znaků LCD

EMAC – Ethernet MAC (verze EMAC Lite). Ovládá na desce obsažený Ethernet PHY (Marvell 88E1111 Giga LAN PHY) a implementuje linkovou vrstvu Ethernetu.

obsahuje systém další systémové součásti:

IRQ controller – řadič přerušení. CPU MicroBlaze umožňuje obsluhovat pouze jediné přerušení. Je-li v systému více periferií pracujících v IRQ módu, je třeba je k CPU připojit přes řadič přerušení.

Timer – systémový časovač. CPU MicroBlaze neobsahuje žádný interní čítač/časovač. Pro Xilkernel nezbytný časovač je nutné do systému začlenit jako externí periferii.

RAM controller – řadič paměti DDR RAM (dual channel) umožňující využít pro aplikace na desce obsaženou externí paměť RAM.

Procesor MicroBlaze je rozšířeno o 8 kB datové a 8 kB instrukční cache (využívá BRAM FPGA). Při běhu programů z RAM tato dramaticky zvyšuje výkon CPU. Z dalších volitelných součástí MicroBlaze je využita 32-bitová HW násobička a barrel shifter. Urychlení násobení a logických/aritmetických posuvů se uplatní zejména při zpracování IP paketů a kódování/dekódování G.711. Procesor je sestaven s neredukovanou pětistupňovou pipeline (optimalizace na rychlost).

[6] V seznamu periferií nejsou uváděny a na schématu zařízení zobrazeny periferie pro ladění – seriová konzole (UART) a JTAG rozhraní.

Schéma platformy je znázorněno na obrázku 2.4.

Jak vyplývá ze seznamu periferií, pro uživatelskou konzoli (rozhraní pro interakci s uživatelem) byla zvolena dvojice zařízení LCD, PS/2 klávesnice. Dvouřádkový šestnáctiznakový displej je pro potřeby interakce uživatele s VoIP telefonem plně dostačující a je integrován na většině pro implementaci využitelných vývojových desek. Na „reálném“ zařízení pak lze také očekávat LCD jako výstupní zařízení.

PS/2 klávesnice je zvolena opět jednak z důvodu své univerzálnosti – většina vývojových desek obsahuje alespoň jeden PS/2 konektor. Navíc je třeba si uvědomit, že SIP adresy jsou ekvivalentem emailových adres, a jedná se tudíž o poměrně značné množství znaků jejichž zadávání pomocí klasických tlačítek (push buttons) by bylo značně nepohodlné. Použít pro vstup seriovou konzoli je přinejlepším ekvivalentní použití samotné klávesnice. U reálného zařízení nelze očekávat plnohodnotnou 104-klávesovou klávesnici, existuje však celá řada „průmyslových klávesnic“, které mají jako rozhraní právě PS/2.

Periferie LCD a PS/2 jsou kompletně (HW+SW) vlastního návrhu, periferie AC97 obsahuje modifikovanou (doplnění obvodu pro reset/inicializaci, viz kapitola 3) HW část z periferie obsažené v referenčním designu k desce ML40x [8] a vlastní SW ovladač. Ostatní použité periferie jsou standardní[7] součástí EDK 9.1.

2.3.2.2 Software

SW platforma zařízení je postavena na OS/realtime kernelu Xilkernel. Jednotlivé funkční bloky zařízení jsou tak řešeny v samostatně běžících, vzájemně komunikujících, vláknech:

console – uživatelská konzole. Vlákno pracuje se vstupy/výstupy od/pro uživatele (klávesnice, LCD). Při událostech na vstupu (nový odchozí hovor, ukončení hovoru, ...) informuje o těchto událostech vlákno SIP stacku – `SIP FSM`.

SIP FSM – implementuje stavový automat SIP stacku. Reaguje na události na síťové vrstvě (příchozí hovor, ukončení hovoru protistranou, ...) a skrze interakci s vláknem `console` na události na uživatelském vstupu. Skrze interakci s vláknem `console` dále řídí uživatelský výstup a skrze interakci s vlákny `RTP send` a `RTP recv` multimediální stream hovoru.

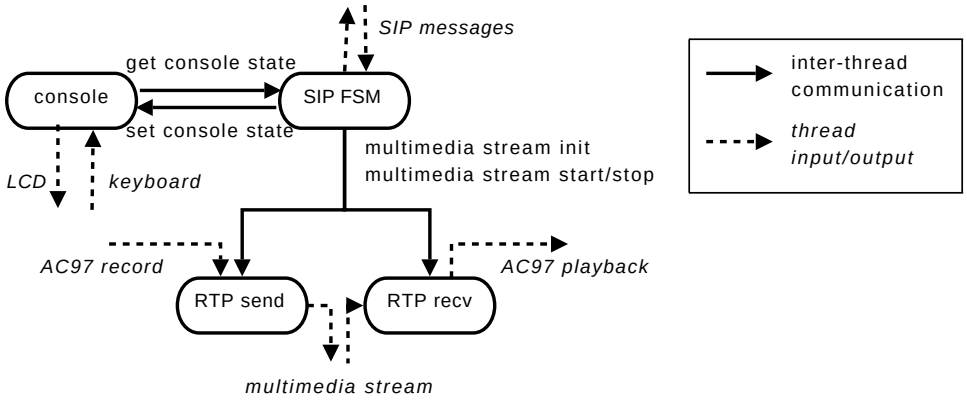
RTP send – Vlákno ovládající odchozí multimediální stream, který generuje z dat získaných z výstupní fronty AC97 kodeku (zvukový vstup – mikrofon). Řízeno vláknem `SIP FSM`.

RTP recv – Vlákno obsluhující příchozí multimediální stream, kterým plní vstupní frontu AC97 kodeku (zvukový výstup – reproduktor). Řízeno vláknem `SIP FSM`.

Schéma vzájemné interakce mezi vlákny a jejich interakci s okolním prostředím (vstupy/výstupy zařízení) zachycuje obr.2.5.

Návrh intuitivně zohledňuje strukturu SIP VoIP stacku oddělením řídícího protokolu SIP a přenosového protokolu RTP do samostatných vláken. Logika RTP je pak ještě dále rozdělena na část zpracovávající příchozí zvukový stream a část generující odchozí zvukový stream. Na rozdíl od SIP protokolu, který pracuje na principu požadavek – odpověď, jsou v protokolu RTP

[7] OPB EMAC Lite je pro volné použití dostupný pouze ve verzi pro vývojáře, jejíž funknost je časově omezena. Pro komerční využití je třeba od firmy Xilinx zakoupit plnohodnotnou verzi IP core.



Obrázek 2.5: Schéma SW platformy zařízení.

odchozí a příchozí data zcela nezávislá (jedná se o samostatné nijak vzájemně nesynchronizované zvukové streamy), proto je rozdělení jejich generování/zpracování rozděleno. Zamezí se tak situacím nebo nutnosti jejich řešení, kdy by jedna operace mohla blokovat druhou, například v důsledku krátkodobého výpadku spojení.

V samostatném vlákně „běží“ také obsluha periferií pro komunikaci s uživatelem či přesněji kompletní logika uživatelského rozhraní. Na rozdíl od interakce mezi vláknem `SIP FSM` a vlákny `RTP send` a `RTP recv`, kde je komunikace jednosměrná a plně deterministická – RTP stream je ovládán (spouštěn/ukončován) výhradně vláknem `SIP FSM` – se stav konzole může měnit jak v důsledku síťové události, tak v důsledku akce uživatele. Při komunikaci mezi vlákny `console` a `SIP FSM` tak potenciálně může dojít k race conditions a následnému deadlocku. Komunikace mezi těmito dvěma vlákny tak v tomto případě musí být ošetřena některým ze synchronizačních prostředků. (Xilkernel dle konfigurace nabízí mutexy, semafory i fronty zpráv).

Aplikační logika – jednotlivá vlákna a jejich propojení – pro svůj běh vyžaduje celou řadu podpůrného kódu – knihoven[8]. Kromě systémových C knihoven, knihoven Xilkernelu a již zmíněné knihovny lwip se jedná zejména o tyto SW komponenty:

keyboard.h, lcd.h – knihovny implementujících obdobu funkcí ze standardní C knihovny `stdio` pro LCD a PS/2 klávesnici jako I/O zařízení.

sip.h – knihovna implementující základní funkce SIP stacku. Jedná se zejména o funkce pro odesílání/příjem SIP/SDP paketů včetně jejich parsování/sestavování a definice příslušných datových struktur pro odesílané/příjimané zprávy.

rtp.h – knihovna poskytující funkce pro odesílání/příjem RTP paketů a odpovídající datové struktury.

timer.h – modul pro podporu časovačů s uživatelsky definovanými časovými limity.

g711.h – funkce implementující dekódování/enkódování zvukových vzorků kompresí G.711 μ-law a A-law.

Funce pro kódování/dekódování zvuku metodami standardu G.711 jsou převzaty z volně šiřitelné referenční implementace firmy Sun Microsystems, ostatní knihovny jsou realizovány v rámci DP.

[8]Pojmem knihovna jsou zde myšleny veškeré SW moduly, které implementují nějakou jinými SW celky využívanou funkcionalitu a nejsou určeny pro samostatný běh jako program/vlákno, nejenom „klasické“ dynamicky linkované knihovny.

Kromě výše uvedených SW komponent vycházejících ze základních požadavků na systém je v realizovaném zařízení implementován modul pro překlad doménových jmen na IP adresy – DNS resolver, který knihovna lwip neobsahuje, a dále modul pro získání IP konfigurace pomocí protokolu DHCP – DHCP klient. Ten je v knihovně lwip 2.0, která je součástí EDK 9.1, implementován, není však oficiálně podporován a k jeho „zprovoznění“ je zapotřebí úprava (patch) knihovny a v samotné aplikaci pak modul pro inicializaci DHCP klienta. Více viz kapitola 3.

3 Realizace

Kapitola *Realizace* podrobně rozebírá všechny části zařízení, které byly v rámci DP implementovány. V části *Řadiče periferií* je popsán realizovaný hardware včetně jeho ovladačů, v dalších sekcích pak všechen software, síťovými stacky počínaje, přes knihovny pro ovládání periferií až po systém propojení všech částí.

3.1 Řadiče periferií

Část věnovaná periferiím zařízení podrobně rozebírá principy fungování, strukturu a rozhraní periferií/řadičů periferií jež byly v rámci realizace zařízení navrženy a implementovány. Podrobně popsán je jak vlastní HW, tak jeho SW vrstva – ovladač – včetně příslušného programátorského rozhraní.

V textu kapitoly není popsán způsob práce s komponentami – návod na jejich začlenění do systému. Všechny periferie jsou zcela standardními EDK komponentami, při jejich přidávání do systému tak lze přímo aplikovat postup popsaný v dokumentaci k EDK, například [3]. Podrobný „step-by-step“ návod pro danou periferii je navíc obsažen v anglické uživatelské dokumentaci modulů v příloze.

Komponenty `lcd_driver` a `ps2_kbd_driver`, a to jak SW tak HW část, jsou kompletně vlastního návrhu, HW část komponenty `opb_ac97_controller` vychází z IP core z referenčního designu pro desky ML40x [8], SW ovladač je kompletně vlastní.

Žádná z těchto tří periferií nemá v EDK 9.1 „oficiální“ řadič (IP core), pro PS/2 a AC97 nicméně existují řadiče obsažené v ukázkových příkladech pro některé vývojové desky. Řadič PS/2 existuje sice včetně SW ovladačů, ale jen pro verzi „dvojitého“ PS/2 portu, který například na desce Spartan-3E starter kit board není, navíc stejně jako referenční design řadiče AC97 potřebuje pro svojí funkci ještě speciální „inicializační“ komponentu (`misc_logic` u referenčního designu k ML40x). Referenční design řadiče AC97 nemá žádný SW ovladač, jen slušně řečeno chaotický příklad použití IP core řadiče.

U PS/2 byl místo úpravy referenčního designu navrhnut kompletně vlastní řadič, který je navržen pro obsluhu jednoho PS/2 portu (ale může být v systému instancován několikrát), nepotřebuje žádnou další komponentu a lze jej „out of the box“ použít jak na desce ML40x, tak na Spartan-3E starter kit board.

U řadiče AC97 byl převzat stávající IP core, který byl doplněn o resetovací/inicializační obvod a SW ovladač. Vznikl tak kompletní, samostatně využitelný modul s plně dokumentovaným programátorským rozhraním.

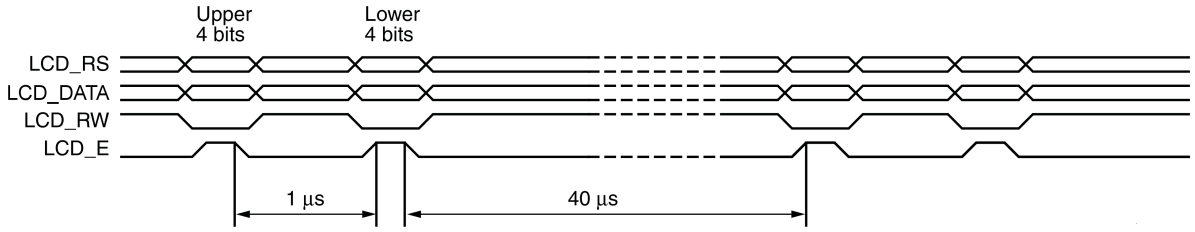
U každého řadiče periferie je uvedena tabulka udávající jím alokované HW a SW zdroje. Informace vztažené na konkrétní chip jsou vázány k chipu XC4VFX12-FF668 na desce ML403. Přehledný podrobný souhrn všech zdrojů/limitů je možné nalézt v podadresáři `report` EDK projektu na přiloženém CD (soubor `system.html`).

3.1.1 LCD

Řadič LCD[1] reprezentovaný komponentou `lcd_driver` realizuje připojení MPU interface kompatibilního znakového LCD k CPU skrze OPB sběrnici.

Řadič obstarává veškerou nízkoúrovňovou komunikaci se zařízením a poskytuje vysokoúrovňové, příkazově orientované, rozhraní k ovládání LCD. Propojení s CPU je realizováno „klasicky“ pomocí registrů mapovaných do paměti. SW ovladač pak poskytuje uživateli funkce pro zasílání dat (jednotlivé znaky) a ovládacích příkazů (smazání zobrazených znaků, posun kurzoru, ...).

[1] Z pohledu procesoru, LCD má také vlastní integrovaný řadič ovládající samotné zobrazování.



Obrázek 3.1: Komunikační protokol LCD. Zdroj: [11].

I/O porty komponenty

Řadič komunikuje s displejem pomocí standardního 4-bitového MPU interface. To sestává z těchto vstupů/výstupů:

LCD_E – read/write enable pulse. Impuls na `LCD_E` indikuje, že jsou na ostatních vstupech platná data a displej je má začít zpracovávat.

LCD_RW – register select. Signál určující, zda jsou aktuální data na `LCD_DATA` příkazem nebo znakem.

LCD_RS – read/write control. Nastavuje LCD do režimu čtení/zápisu.

LCD_DATA[3:0] – data. Čtyřbitový datový vektor pro znak/příkaz.

Jednotlivé I/O porty komponenty (IP core) `lcd_driver` mají názvy totožné s názvy signálů interface. Vlastní LCD má díky podpoře 8-bitového režimu osmibitový datový vstup `LCD_DATA`, ve 4-bitovém režimu se tak využívají pouze piny [7:4].

Při připojování řadiče do systému, je kromě připojení I/O portů a signálů OPB sběrnice nutné v nastavení IP core ještě nastavit konstantu `C_OPB_Clk_FREQ_HZ` udávající zařízení frekvenci OPB sběrnice systému.

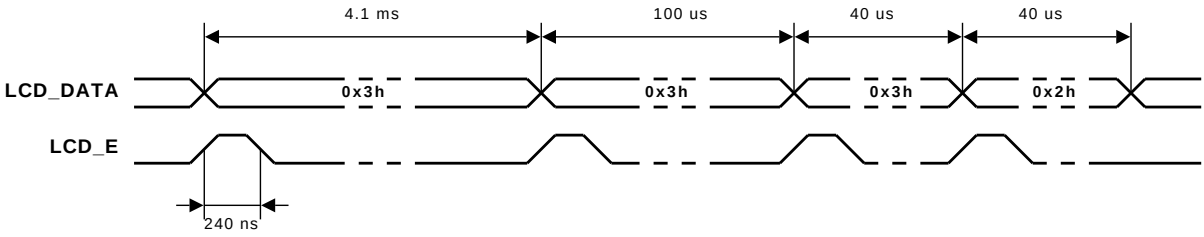
Princip činnosti, popis HW

Princip přenosu přes MPU interface je prostý. V prvním kroku se dle typu požadované operace (zápis/čtení, příkaz/data) nastaví signály `LCD_RW`, `LCD_RS` a případně data pro zápis na `LCD_DATA`. Ve druhém kroku se pak pomocí impulsu na `LCD_E` zahájí vlastní přenos.

Příkazy/data jsou osmibitová, ve čtyřbitovém módu je proto třeba přenášet je po čtyřbitových částech – nibblech. Tento přenos je v řadiči plně HW záležitostí, SW ovladač pracuje transparentně s 8-bitovými příkazy/znaky. Protokol komunikace se zařízením je znázorněn na obrázku 3.1.

Kromě obsluhy komunikace s displejem má HW řadič ještě druhou základní funkci – provádí jeho inicializaci. Displej je před použitím třeba inicializovat pro 4-bitový mód přenosu, bez této inicializace je zcela nefunkční. Inicializace se provádí speciální sekvencí příkazů, která je zobrazena na obrázku 3.1. Inicializace displeje je provedena automaticky po připojení řadiče k napájecímu napětí, tedy po zapnutí systému.

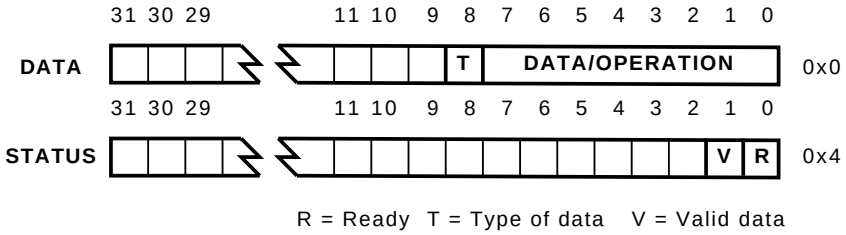
Přestože LCD je vstupně/výstupní zařízení – umožňuje číst obsah paměti znaků a získat stav aktuálního přenosu – řadič k němu přistupuje pouze jako k čistě výstupnímu zařízení. Operace "Read Busy Flag and Address" a "Read Data from CG RAM or DD RAM" tak nejsou řadičem podporovány. To ovšem není příliš velké omezení, neboť stav přenosu může být, a v řadiči také je, určován na základě maximání prováděcí doby jednotlivých operací. Také počáteční obsah paměti LCD je znám a aplikace tak může aktuální obsah určit na základě



Obrázek 3.2: Inicializační sekvence LCD.

provedených operací. Ve skutečnosti proto drtivá většina aplikací pracuje s LCD jako s čistě výstupním zařízením.

Činnost řadiče je ovládána skrze dva 32-bitové registry – první, označený jako datový (data register) je registr sloužící pro zápis požadovaného příkazu/znaku. Z tohoto registru není možné data číst, lze do něj pouze zapisovat. Kromě kódu samotného je v bitu číslo 8 uložena informace, zda data reprezentují znak či příkaz. Druhý, stavový registr (status register), slouží k ovládání a indikaci stavu přenosu. „1“ v bitu č. 0 indikuje, že displej je připraven přijmout znak/příkaz. Pokud je v bitu č. 1 hodnota „1“, jsou data v datovém registru považována za platná a řadič spustí jejich zápis do LCD.



Obrázek 3.3: Registry LCD řadiče.

Programátorský model

Ovladač LCD řadiče reprezentovaný hlavičkovým souborem `lcd_driver.h` implementuje funkce pro základní komunikaci s displejem skrze jeho rozhraní. Poskytuje pouze funkce realizující přenos znaků/operací na displej a umožňuje tak sám o sobě s displejem realizovat pouze operace dané možnostmi daného HW. Případnou „vyšší logiku“ ovládání (výpis řetězců, různé druhy posunu obsahu, ...) je nutné případně realizovat v knihovně vystavěné nad ovladačem.

SYNOPSE

```
#include <lcd_driver.h>

void lcd_init(LCDInst *instance, Xuint32 baseaddr)
void lcd_write_cmd(LCDInst *instance, Xuint8 cmd)
void lcd_write_data(LCDInst *instance, Xuint8 data)
```

POPIS

Funkce `lcd_init()` inicializuje instanci zařízení (systém může obsahovat více LCD) danou jeho počáteční adresou v adresním prostoru CPU (base address). Funkce musí být zavolána před prvním použitím periferie.

Resource Type	Used	Available	Percent
Slices	179	5472	3
Slice Flip Flops	191	10944	1
4 input LUTs	336	10944	3
IOs	7	320	2
SW driver size			3352 B

Tabulka 3.1: Řadič LCD - alokované zdroje.

`lcd_write_cmd()` pošle na displej instrukci s kódem `cmd`. Kompletní seznam možných instrukcí včetně jejich popisu udává datasheet daného displeje. Velice přehledný popis lze také nalézt v dokumentaci k některým vývojovým deskám firmy Xilinx, například [11]. Hlavičkový soubor ovladače také obsahuje definice symbolických konstant nejdůležitějších operací dostupných na displejích integrovaných na vývojových deskách firmy Xilinx.

Funkce `lcd_write_data()` zašle na displej data (znak), která mají být zapsána do CG RAM nebo DD RAM (záleží na předchozích instrukcích) displeje (respektivě jeho integrovaného řadiče).

Pokud je ovladač zkompilován s definicí konstanty `INCLUDE_LCD_SELFTEST` (hlavičkový soubor `lcd_driver.h`), je kromě výše uvedených dostupná ještě funkce `lcd_selftest()`, která slouží k jednoduchému otestování funkčnosti displeje a jeho propojení se systémem.

3.1.2 PS/2

Komponenta `ps2_kbd_driver` představuje řadič PS/2 klávesnice/myši[2]. Řadič obstarává nízkoúrovňovou PS/2 komunikaci a poskytuje uživateli znakově orientované rozhraní pro obousměrnou komunikaci s připojeným zařízením.

Řadič umožňuje provoz ve dvou základních módech – polling mode (periodické testování stavu zařízení) a interrupt mode (přerušení generované řadičem).

I/O porty komponenty

I/O porty komponenty tvoří dva[3] I/O signály – `PS2_clk` a `PS2_data`, které odpovídají příslušným vodičům PS/2 portu (konektoru).

Mimo porty a standardní signály OPB sběrnice poskytuje komponenta ještě signál přerušení `IP2INTC_Irpt`. Ten je v případě, že má komponenta pracovat v režimu přerušení, nutné zapojit na příslušný vstup CPU, či řadiče přerušení je-li použit.

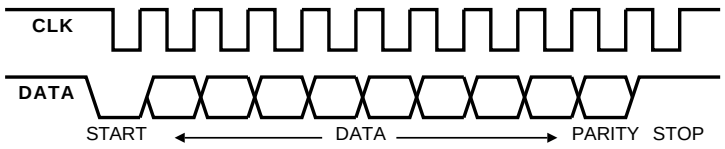
Při připojování řadiče do systému, je kromě připojení I/O portů a signálů OPB sběrnice nutné v nastavení IP core ještě nastavit konstantu `C_OPB_Clk_FREQ_HZ` udávající zařízení frekvenci OPB sběrnice systému.

Princip činnosti, popis HW

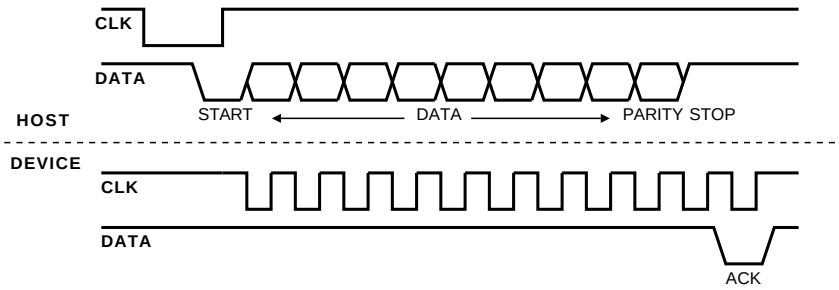
PS/2 je obousměrný synchronní seriový protokol navržený pro dvoubodové připojení periferií k PC. Periferie jsou v terminologii PS/2 označovány jako zařízení (device), počítač pak jako hostitel (host). Jejich fyzické propojení je (kromě napájení periferie) realizováno dvěma vodiči –

[2]Komponenta byla primárně navržena pro ovládání PS/2 klávesnice, proto "kbd_driver". Protože ale HW ani SW neimplementuje žádnou funkcionalitu specifickou pro klávesnici, ale pouze obecné komunikační rozhraní, je pomocí této periferie stejně dobře možné ovládat i PS/2 myš.

[3]Jelikož se jedná o vstupně-výstupní porty, je každý interně reprezentován třemi signály – vstupem *_I, výstupem *_O a signálem řídícím třístavový budič *_T. V prostředí EDK jsou však reprezentovány jako jediný port typu IO (input-output).



Obrázek 3.4: Protokol PS/2, přenos periferie-hostitel.



Obrázek 3.5: Protokol PS/2, přenos hostitel-periferie.

datovým (data) a hodinovým (clock). Vzhledem k tomu, že oba dva vodiče mohou být ovládány oběma stranami spojení, jsou koncové obvody realizovány zapojením s otevřeným kolektorem.

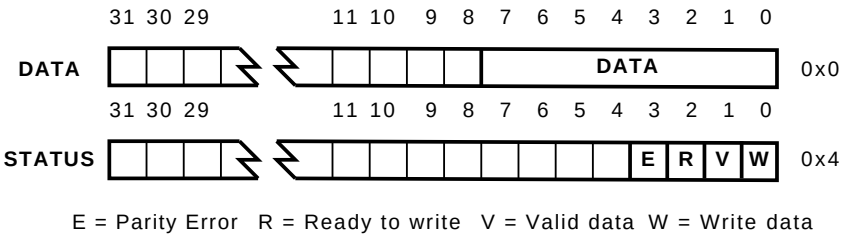
Spojení je nečinné, pokud jsou oba vodiče ve stavu logická „1“. Pouze v tomto případě může začít periferie vysílat data. Hostitel kontroluje sběrnici a může ve kterémkoliv okamžiku přenos dat ze zařízení přerušit nastavením signálu clock na „0“. Hodinové pulzy jsou vždy generovány periferií. Pokud chce hostitel vysílat data, musí nastavit clock na „0“, následně na „0“ nastavit signál data a uvolnit clock. Periferie, která neustále sleduje stav obou vodičů, poté začne generovat hodinový signál, který umožní hostiteli odvysílat požadovaná data.

Vlastní přenos probíhá v 11-12 bitových sekvencích (záleží na směru přenosu, viz dále), přenášeno je 8 datových bitů a jejich lichá parita. Přenos je uvozen start bitem („0“) a ukončen stop bitem („1“), v případě přenosu hostitel-periferie za stop bitem ještě následuje potvrzení příjmu – ack bit („0“) – nastavený periferií. Data přenášená ve směru periferie-hostitel jsou čtena při sestupné hraně hodin, data přenášená ve směru hostitel-periferie naopak při hraně vzestupné. Grafické znázornění obou typů přenosů uvádí obrázky 3.4 a 3.5. Frekvence hodinového signálu se může pohybovat v rozmezí 10-16,7 kHz.

Činnost řadiče je ovládána skrze dva 32-bitové registry – první, označený jako datový (data register), je registr sloužící pro zápis požadovaného znaku v případě komunikace hostitel-periferie, nebo čtení přijatého znaku v případě komunikace periferie-hostitel. Druhý, stavový registr (status register), slouží k ovládání a indikaci stavu přenosu. Zápis/čtení funguje dle principu protokolu – operace zápis má přednost před operací čtení. Konkrétně to znamená, že je-li nastaven bit č. 0 stavového registru na „1“, je přerušen případný příjem znaku a je zahájeno vysílání. Bit č. 1 ve stavu logická „1“ značí, že v datovém registru je dostupný přijatý znak. Bit je řadičem nulován vždy při začátku příjmu nového znaku. Bit č. 2 nastavený na „1“ signalizuje, že je řadič připraven vysílat data (předchozí operace byla dokončena) a konečně bit č. 3 indikuje případnou chybu parity.

Programátorský model

Ovladač PS/2 řadiče reprezentovaný hlavičkovým souborem `ps2_kbd_driver.h` implementuje základní interface pro práci s PS/2 periferií. Poskytuje funkce pro obousměrný přenos znaků



Obrázek 3.6: Registry PS/2 řadiče.

z/do zařízení a funkce pro obsluhu periferií generovaného přerušení.

SYNOPSE

```
#include <ps2_kbd_driver.h>

void ps2_kbd_init(PS2KbdInst *instance, Xuint32 baseaddr, PS2KbdMode opmode)
void ps2_kbd_set_irq_handler(PS2KbdInst *instance, ps2_kbd_irq_handler hndlr)
Xuint8 ps2_kbd_get_code(PS2KbdInst *instance)
void ps2_kbd_set_code(PS2KbdInst *instance, Xuint8 code)
void ps2_kbd_irq_handler(void *instance)
```

POPIS

Funkce `ps2_kbd_init()` inicializuje instanci ovladače pro periferii danou její počáteční adresou v adresním prostoru CPU (base address). Parametr `opmode` udává, v jakém režimu bude periferie pracovat. Možné volby jsou `PS2_KBD_POLLING_MODE` a `PS2_KBD_IRQ_MODE`.

Funkce `ps2_kbd_get_code()` vrací poslední přijatý znak zaslaný periferií, pokud byl od posledního volání funkce přijat nový znak nebo pokud v IRQ režimu není interní fronta znaků prázdná (viz funkce `ps2_kbd_set_irq_handler()`), 0 v opačném případě. Pokud při přenosu došlo k chybě (nesouhlasící parita), funkce zašle periferii příkaz pro opakovaní posledního znaku (0xFE) a vrátí 0.

`ps2_kbd_set_code()` je funkce sloužící k přenosu znaků od hostitele k periferii. Volání této funkce přeruší případný přenos znaku z periferie, pokud stále probíhá dříve inicializovaný přenos z hostitele k periferii, je do ukončení předchozího přenosu aktuální požadavek zablokován.

Pokud je periferie nastavena pro práci v režimu přerušení, jsou dostupné navíc funkce `ps2_kbd_set_irq_handler()` a `ps2_kbd_irq_handler()`.

`ps2_kbd_set_irq_handler()` nastavuje uživatelskou obslužnou rutinu, která je v případě přerušení volána. Ovladač implementuje vlastní FIFO o velikosti 32 znaků[4], do kterého jsou v interrupt modu přijaté znaky ukládány. Při volání funkce `ps2_kbd_get_code()` jsou potom znaky čteny z této fronty. Pokud je třeba zpracovávat znaky přímo v přerušovací rutině, je možné volat `ps2_kbd_get_code()` v jejím těle – aktuálně přijatý znak už je v okamžiku volání uživatelské obslužné rutiny do fronty zařazen.

`ps2_kbd_irq_handler()` je funkce, která musí být v systému nastavena jako obslužná funkce pro přerušení z PS/2 řadiče. Nastavuje interní mechanismy ovladače a volá funkcí `ps2_kbd_set_irq_handler()` definovanou obslužnou rutinu.

Pokud je ovladač zkompilován s definovanou konstantou `INCLUDE_PS2_KBD_SELFTEST`, (soubor `ps2_kbd_driver.h`) je kromě výše uvedených dostupná ještě funkce `kbd_selftest()`, která

[4]Velikost fronty lze nastavit v hlavičkovém souboru ps2_kbd_driver.h.

Resource Type	Used	Available	Percent
Slices	129	5472	2
Slice Flip Flops	118	10944	1
4 input LUTs	196	10944	1
IOs	2	320	1
SW driver size			6164 B

Tabulka 3.2: Řadič PS/2 - alokované zdroje.

slouží k jednoduchému otestování funkčnosti řadiče a jeho propojení se systémem.[5]

3.1.3 AC97

Komponenta `opb_ac97_controller` je IP core realizující propojení CPU se zvukovým kodekem kompatibilním se standardem AC97[1]. Řadič obstarává komunikaci s kodekem přes rozhraní kodeku – AC-link a umožňuje přístup k funkcím kodeku skrze registry mapované přes sběrnici OPB do adresního prostoru CPU.

Řadič obsahuje pro oba směry datového toku (záznam/přehrávání) vyrovnávací paměť (FIFO) o velikosti 16 vzorků a umožňuje provoz ve dvou základních módech – polling mode a interrupt mode. Nastavení režimu je pro záznam a přehrávání nezávislé. Pracuje-li řadič v režimu přerušení, je toto generováno „událostmi“ v obou frontách. Vyvolání přerušení je možné při plné/prázdné frontě i při frontě „poloplné“ a „poloprázdné“.

I/O porty komponenty

I/O porty komponenty – `Bit_Clk`, `Sync`, `SData_Out`, `SData_In` a `Reset` odpovídají příslušným portům AC97 kodeku pro připojení řadiče. Vodiče `Playback_interrupt` a `Record_interrupt` pak představují signály přerušení pro přehrávání/záznam.

Při připojování řadiče do systému, je kromě připojení I/O portů a signálů OPB sběrnice nutné v nastavení IP core ještě nastavit konstantu `C_OPB_Clk_FREQ_HZ` udávající zařízení frekvenci OPB sběrnice systému a konstanty `C_PLAY_INTR_LEVEL` a `C_REC_INTR_LEVEL`. Tyto konstanty slouží k nastavení módu generování přerušení a mohou mít následující hodnotu:

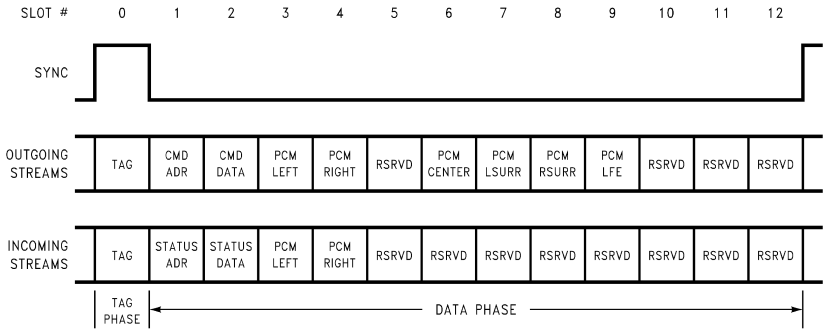
0 – bez generování přerušení.
1 – přerušení při prázdné frontě (0 položek).
2 – přerušení při „poloprázdné“ frontě (0-7 položek).
3 – přerušení při „poloplné“ frontě (8-16 položek).
4 – přerušení při plné frontě (16 položek).

Dále je možné pomocí nastavení jak část pro záznam, tak část pro přehrávání z řadiče zcela odstranit nastavením konstant `C_PLAYBACK` či `C_RECORD` na „0“.

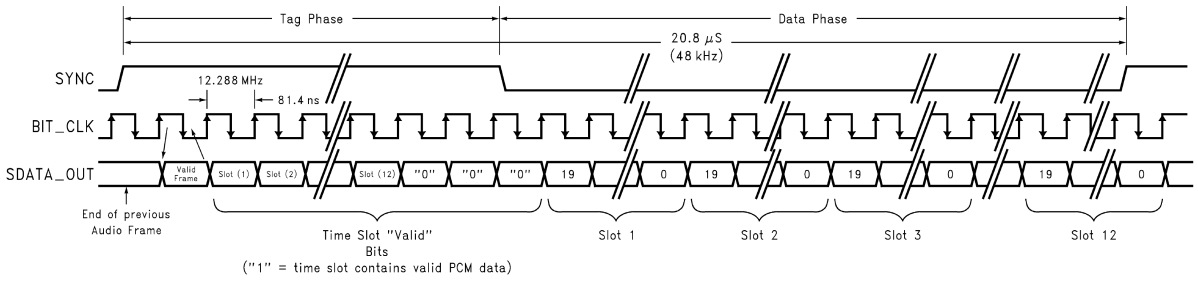
Princip činnosti, popis HW

Řadič komunikuje s kodekem skrze sériové rozhraní nazývané AC-link. Protokol AC-link je postaven na přenosu rámců sestávajících vždy z dvanácti po sobě jdoucích 20-bitových datových slotů, uvozených jedním slotem úvodním (tag). Obsah jednotlivých slotů a formát výstupního (vstupní je ekvivalentní) rámce zobrazují obrázky 3.7 a 3.8.

[5] Funkce je specifická pro a vyžaduje připojenou PS/2 klávesnici.



Obrázek 3.7: Formát rámce AC97 kodeku. Zdroj: [5].



Obrázek 3.8: Výstupní rámec AC97 kodeku. Zdroj: [5].

V jednotlivých slotech jsou kromě vlastních zvukových vzorků přenášeny také požadavky na čtení/zápis do konfiguračních registrů kodeku a odpovědi na ně. Podrobný popis komunikace i seznam registrů včetně jejich podrobného popisu uvádí specifikace [1]. Řadič však nepodporuje kompletní možnosti kodeku – podporován například není přenos dat z jiných než levého a pravého kanálu a rozlišení (bit depth) vzorků je pouze 16 bitů, přestože kodeky podporují rozlišení až 20-bitové.

Kromě řízení komunikace s kodekem řadič obstarává i jeho inicializaci/„studený“ reset (cold reset). Signál vyvolávající reset je připojen na reset signál OPB sběrnice, při zresetování systému tak dojde i k zresetování AC97 kodeku do výchozího nastavení.

Činnost řadiče je ovládána skrze 7 32-bitových registrů:

INFIFO – registr pro ukládání dat do vstupní fronty kodeku (přehrávání).

OUTFIFO – registr pro ukládání dat do výstupní fronty kodeku (záznam).

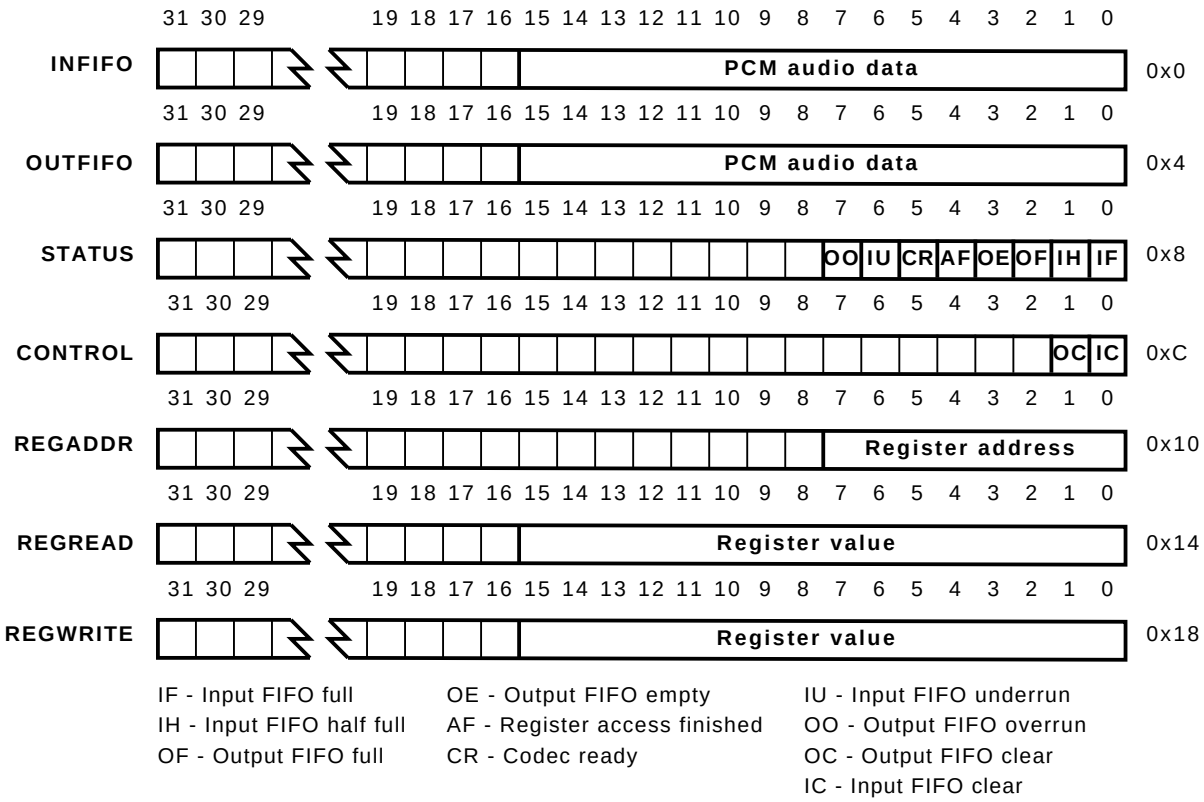
STATUS – stavový registr udávající aktuální stav obou front a kodeku vůbec.

CONTROL – registr pro kontrolu (vyprázdnění) front.

REGADDR – registr pro uložení adresy registru kodeku ze/do kterého se má zapisovat/číst.

REGREAD – obsahuje hodnotu, která byla přečtena z registru `REGADDR`.

REGWRITE – registr pro uložení hodnoty, která má být zapsána do registru `REGADDR`.



Obrázek 3.9: Registry řadiče AC97.

Programátorský model

Ovladač řadiče AC97 reprezentovaný hlavičkovým souborem `opb_ac97_controller.h` implementuje základní množinu funkcí pro práci s AC97 kodekem. Jedná se zejména o funkce pro práci s oběma FIFO, funkce pro obsluhu přerušení a dále pak podpora některých možností nastavení kodeku přes konfigurační registry (hlasitosti, vzorkovací frekvence, vstupní zařízení).

SYNOPSE

```
#include <opb_ac97_controller.h>

void ac97_init(AC97Inst *instance, Xuint32 baseaddr);
void ac97_set_playback_handler(AC97Inst *instance, AC97_handler handler);
void ac97_set_record_handler(AC97Inst *instance, AC97_handler handler);
void ac97_playback_irq_handler(void *instance);
void ac97_record_irq_handler(void *instance);
void ac97_set_sample_rates(AC97Inst *instance, Xuint32 DAC, Xuint32 ADC);
void ac97_set_volume(AC97Inst *instance, Xuint32 reg, Xuint32 value);
void ac97_set_record_src(AC97Inst *instance, Xuint32 value);
void ac97_set_record_gain(AC97Inst *instance, Xuint32 gain);
int ac97_write(AC97Inst *instance, Xuint32 *letf, Xuint32 *right);
int ac97_read(AC97Inst *instance, Xuint32 *letf, Xuint32 *right);
```

Resource Type	Used	Available	Percent
Slices	146	5472	2
Slice Flip Flops	190	10944	1
4 input LUTs	205	10944	1
IOs	5	320	2
SW driver size			7300 B

Tabulka 3.3: Řadič AC97 - alokované zdroje.

POPIS

Funkce `ac97_init()` inicializuje zařízení `instance` s bázovou adresou `baseaddr`. Tato funkce musí být zavolána před jakoukoliv prací s příslušnou instancí zařízení.

`ac97_write()` je funkce sloužící k zapsání jednoho, respektive dvou zvukových vzorků do vstupní fronty řadiče. Parametr `left` určuje PCM data levého kanálu, parametr `right` PCM data pravého kanálu. Funkce vrací 1, pokud byla operace provedena, 0 v případě, že je vstupní fronta plná.

Funkce `ac97_read()` získá aktuální položku (zvukový vzorek) z výstupní fronty řadiče. Pokud fronta není prázdná, vrací 1, v opačném případě 0. `left` je 16-bitový PCM vzorek z levého kanálu kodeku, `right` 16-bitový PCM vzorek z pravého kanálu.

Funkce `ac97_set_playback_handler()` a `ac97_set_record_handler()` slouží k nastavení uživatelských obslužných rutin pro daný typ přerušení. Zaregistrované funkce jsou volány ve funkcích ovladače pro obsluhu přerušení `ac97_playback_irq_handler()` a `ac97_record_irq_handler()`, které musí být zaregistrovány v systému jako obslužné rutiny odpovídajících přerušení.

Funkce `ac97_set_sample_rates()` slouží k nastavení vzorkovací frekvence kodeku. Výchozím nastavením kodeku je fixní vzorkovací frekvence 48 kHz, zavoláním této funkce dojde k přepnutí kodeku do VRA (Variable Rate Audio) módu a nastavení vzorkovací frekvence ADC na hodnotu `ADC` a DAC na hodnotu `DAC`. Hodnoty `DAC` a `ADC` odpovídají vzorkovací frekvenci v Hz.

K nastavení hlasitosti jednotlivých výstupů[6] kodeku slouží funkce `ac97_set_volume()`. Konstanty pro adresy registrů `reg` ovládající jednotlivé výstupy jsou uvedeny v hlavičkovém souboru. Bity 0-4 hodnoty `value` určují hlasitost pravého kanálu výstupu, bity 8-12 hlasitost levého kanálu. Nejvyšší bit hlasitosti kanálu určuje jeho ztlumení, pokud je nastaven bit registru č. 15 na „1“, je výstup zcela vypnut (mute). Hodnoty pro hlasitost jednotlivých kanálů nastavují ztišení výstupu, maximální hlasitost vstupu tedy má hodnotu 0x00 a minimální 0xFF. Hlavičkový soubor definuje některé konstanty pro nastavení hlasitosti, například `AC97_VOL_MUTE` (0x8000) či `AC97_VOL_MAX` (0x0000). Podrobnosti o nastavení hlasitosti lze najít v [1].

Funkce `ac97_set_record_gain()` nastavuje zesílení vstupních signálů (mikrofon, line-in). Princip nastavení je tedy opačný, než u nastavení hlasitosti – maximální hodnota zesílení kanálu je 0xFF, minimální 0x00.

Funkce `ac97_set_record_src()` nastavuje zdroj signálu pro ADC kodeku. V hlavičkovém souboru jsou definovány konstanty pro vstup mikrofonu a line-in vstup – `AC97_REC_MIC`, `AC97_REC_LINEIN`.

[6] Výstupem jsou i audio vstupy, neboť tyto jsou interně propojeny s výstupy.

3.2 Síťová komunikace

Kapitola *Síťová komunikace* podrobně popisuje implementace knihoven/modulů (označovaných souhrně jako „stack“), které v zařízení obstarávají síťovou komunikaci, tedy sestavení hovoru a přenos audio dat. U každého modulu je kromě vlastního popisu implementace vždy rozebrána příslušná část specifikace síťového protokolu, ze které implementace vychází.

3.2.1 SIP/SDP stack

SIP/SDP stack obsluhuje tu část komunikace zařízení, kdy je sestavován hovor (SIP) a protistranami domlouvány jeho parametry (SDP). Zcela obecný úvod do protokolu je v kapitole 2.1.2, následující popis protokolu se tak zabývá již přímo popisem zpráv protokolu a průběhem komunikace mezi dvěma SIP klienty. Popis protokolu nezachází do úplných detailů, ty lze nalézt v příslušné specifikaci protokolu, měl by však být dostatečný k pochopení, jak funguje implementovaný SIP/SDP stack.

Popis SIP protokolu

Jak se uvádí i přímo ve specifikaci protokolu [14], vychází protokol SIP z protokolu HTTP a má tudíž podobný model výměny zpráv i jejich formát. Komunikace probíhá systémem klient-server, nebo také požadavek-odpověď, přičemž obě dvě strany mohou v rámci jednoho „hovoru“ vystupovat zároveň jako klient a zároveň jako server. Rozdělení je čistě „logické“, interně má každý SIP klient (zařízení) jak část klientskou, ve specifikaci nazývanou UAC – User Agent Client, tak část serverovou, ve specifikaci nazývanou UAS – User Agent Server. UAC obstarává generování požadavků (sestavení nového hovoru, ukončení aktuálního hovoru, ...), UAS pak zpracovává příchozí požadavky a generuje na ně odpovědi (příjem žádosti o ukončení aktuálního hovoru od protistrany a její potvrzení, odmítnutí příchozího hovoru patřičným chybovým kódem, ...).

Dalšími důležitými pojmy pro pochopení komunikačního protokolu jsou dialog a transakce. Dialog reprezentuje veškerou SIP komunikaci v rámci jednoho „hovoru“. Dialog se pak skládá z několika transakcí. Samostatnými transakcemi jsou například navázání spojení a jeho ukončení. Obě dvě transakce ale náleží jednomu dialogu.

Pojmem session je pak nazýváno vlastní multimediální spojení, přesně je session definována citací z SDP specifikace (RFC 4566[12]) jako: *”množina vysílačů a příjmačů multimediálních dat a datových proudů mezi těmito vysílači”*.

Pro přenos zpráv je využíván protokol UDP[7], standardní port je port 5060.

Zprávy protokolu jsou textové a mají formát daný následující BNF gramatikou:

```
generic-message  =  start-line
                    *message-header
                    CRLF
                    [ message-body ]
start-line       =  Request-Line / Status-Line
```

Za úvodní řádkou (`start-line`) následují hlavičky zprávy a prázdná řádka oddělující úvodní řádku a hlavičky od případného těla zprávy. Úvodní řádka požadavku má formát:

```
Request-Line  =  Method SP Request-URI SP SIP-Version CRLF
```

[7] Standard specifikuje i TCP variantu přenosu, která je však v praxi využívána minimálně. Realizované zařízení komunikaci pomocí protokolu TCP nepodporuje.

Úvodní řádka odpovědi pak:

```
Status-Line  =  SIP-Version SP Status-Code SP Reason-Phrase CRLF
```

kde SP je oddělující mezera a CRLF dvouznakový konec řádku známý z OS Windows. Metoda dotazu (`Method`) může být jedna ze šestice:

REGISTER – metoda sloužící k zaregistrování klienta na registračním serveru. Tato funkcionalita není zařízením podporována a nebude již dále diskutována.

INVITE – metoda sloužící k vytvoření session.

ACK – metoda sloužící pro potvrzení vytvořené session.

CANCEL – slouží k ukončení session v sestavovací fázi.

BYE – metoda sloužící k ukončení probíhající SIP session.

OPTIONS – slouží k zjišťování možností SIP serverů.

Položka `Request-URI` udává SIP URI (adresu) příjemce paketu. V odpovědi za použitou verzí SIP protokolu (SIP/2.0) následuje stavový kód odpovědi (`Status-Code`) a jeho textový popis (`Reason-Phrase`). Stavové kódy jsou obdobné jako u protokolu HTTP – jsou trojciferné a první číslice udává typ kódu (1xx – provizorní kódy, 2xx – úspěch, 4xx – chyba).

Hlavičky zprávy mají obecně formát:

```
header  =  "header-name" HCOLON header-value *(COMMA header-value)
```

Specifikace definuje více než 40 hlaviček, implementace však umí pracovat pouze s těmi nejnutnějšími (povinnými), ostatní ignoruje. Povinné hlavičky zprávy jsou:

Via – udává typ transportu a transportní cestu zprávy. Každý SIP element v cestě zprávy přidává do zprávy tuto hlavičku se svojí adresou. Na adresu uvedenou v této hlavičce směřují odpovědi na zprávu. Součástí této hlavičky musí být povinný parametr `branch` jednoznačně identifikující spojení.

To – udává „logickou“ adresu příjemce požadavku. Tato nemusí souhlasit s aktuální „fyzickou“ adresou klienta. Součástí hlavičky je povinný parametr `tag` identifikující spojení, který nastavuje příjemce.

From – udává „logickou“ adresu iniciátora dotazu. Součástí hlavičky je povinný parametr `tag`, který nastavuje odesílatel.

Call-ID – unikátní identifikátor SIP dialogu.

CSeq – identifikátor transakce dialogu. Každá ze stran dialogu spravuje vlastní identifikátor, jehož sekvenční číslo s každou další touto stranou iniciovanou transakcí o 1 zvětšuje.

Contact – udává aktuální umístění („fyzickou“ adresu) klienta. Hlavička je povinná, pokud pomocí dané zprávy může dojít k sestavení dialogu.

Content-Length – udává velikost těla zprávy v bytech. Povinná pokud je velikost 0 (prázdné tělo).

```
INVITE sip:user@pc.localnet SIP/2.0
Via: SIP/2.0/UDP 192.168.1.3;rport;branch=z9hG4bK1452956
Max-Forwards: 70
To: <sip:user@pc.localnet>
From: <sip:user@fpga.localnet>;tag=56467
Call-ID: 3453454@fpga.localnet
CSeq: 723 INVITE
Contact: <sip:192.168.1.3>
Allow: INVITE,ACK,BYE,CANCEL,OPTIONS
Content-Type: application/sdp

v=0
o=- 384189142 727047710 IN IP4 192.168.1.3
s=-
c=IN IP4 192.168.1.3
t=0 0
m=audio 8000 RTP/AVP 8 0
```

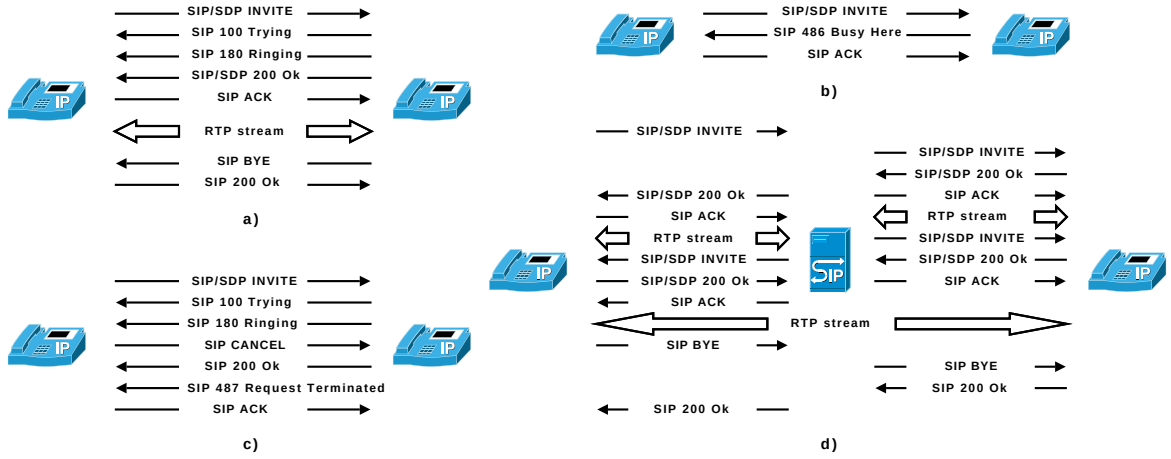
Obrázek 3.10: Formát SIP/SDP zprávy.

Content-Type – určuje typ těla zprávy. Hlavička je povinná, pokud zpráva obsahuje tělo.

Max-Forwards – udává maximální počet SIP elementů, přes které může zpráva putovat, než bude zahozena. Každý SIP element v cestě toto číslo sníží o jedna.

Tělo zprávy může obecně obsahovat data libovolného protokolu popisujícího nějaký druh session, pro potřeby VoIP se však používá výhradně protokol SDP. V případě, že zpráva obsahuje SDP tělo, musí být ve zprávě uvedena příslušná hlavička udávající jeho typ:
`Content-Type: application/SDP`. Formát SDP těla zprávy je popsán v samostatné části věnující se SDP. Příklad SIP zprávy je na obrázku 3.10.
Komunikace mezi SIP klienty při VoIP hovoru má obecně následující průběh:

1. Volající inicializuje spojení a otevře dialog pomocí INVITE dotazu. INVITE zpráva je současně i první transakcí dialogu. Součástí INVITE zprávy je SDP popis session(s), kterou volající volanému nabízí.
2. Volaný na INVITE může odpovědět některým z provizorních stavových kódů, například 180 Ringing – „vyzváním“.
3. Pokud se volající rozhodne ukončit pokus o spojení dříve, než obdrží „definitivní“ odpověď (kód 200 nebo 4xx), může tak učinit zasláním zprávy CANCEL. Volaný pak „stornování“ hovoru potvrdí odpovědí 200 Ok (S metodou CSeq – CANCEL) a navíc pošle informaci o ukončení hovoru – 487 Request Terminated (viz obr. 3.11 c).
4. Po případných provizorních kódech následuje „definitivní“ odpověď volaného. Ta může být dvojího typu – hovor je buď přijat, pak volaný odpoví kódem 200 Ok, nebo je hovor z nějakého důvodu odmítnut (obsazeno, nabízeny pouze nepodporované kodeky, ...) či přesměrován, pak volaný odpovídá některým z chybových kódů (4xx, 6xx) nebo informací o přesměrování (zpráva 3xx). Součástí případné kladné odpovědi je SDP popis session volaného.
5. Volající potvrdí příjem odpovědi zprávou ACK. Pkud byl odpovědí chybový kód, je tímto dialog a tím i VoIP hovor ukončen. V případě odpovědi 200 Ok je potvrzením ukončena transakce, SIP dialog dále pokračuje a jsou „odstartovány“ dohodnuté session.



Obrázek 3.11: Základní průběhy SIP dialogu.

6. V rámci dialogu mohou probíhat různé transakce. Při přímém spojení je obvykle druhou a zároveň poslední transakcí až ukončení dialogu. Pokud je volajícím SIP server, dochází v této fázi například pomocí mechanismu nazývaného re-invite k přesměrování session na skutečného volajícího (viz obr. 3.11 d).
7. Ukončení dilogu a tím i celého VoIP hovoru je iniciováno transakcí začínající zprávou BYE, odeslanou libovolnou stranou dialogu. Protistrana na BYE zprávu odpovídá odpovědí se stavovým kódem 200 Ok, čímž je dialog ukončen.

Základní schémata komunikace jsou zobrazena na obrázku 3.11. Obrázek a) zobrazuje úspěšné spojení dvou SIP klientů, obrázek b) odmítnuté spojení. Na obrázku c) je situace, kdy se klient rozhodne zrušit právě probíhající INVITE transakci. Konečně obrázek d) ukazuje situaci, kdy v rámci aktivního dialogu dojde ke změně již vytvořené session, tzv. re-INVITE, které používají SIP servery pracující v proxy módu.

Ze zařízení podporovaných schémat spojení není na obrázku 3.11 zobrazeno spojení klientů skrze SIP server pracující v redirect módu. Takové spojení je posloupností obrázků b) a a). Původní hovor je serverem odmítnut zprávou 3xx (přesměrování) – obr. b), a v odpovědi je volajícímu sdělena aktuální adresa volaného, na kterou poté směřuje nový pokus o spojení (INVITE požadavek) dle základního schématu a).

Jelikož veškerá komunikace je prováděna nezabezpečeným[8] komunikačním kanálem, obsahuje specifikace mechanismy umožňující chyby přenosu detekovat a ošetřit. Kromě výše popsaného identifikátoru transakce, který obsahuje i sekvenční číslo, se jedná zejména o pravidla pro práci s timeouty a duplicitnými zprávami. Pokud dorazí duplicitní zpráva, odpovídá na ni klient stejnou odpovědí jako na první takovou zprávu. Pokud v časovém limitu nedorazí odpověď na poslední odeslanou zprávu, je zpráva vyslána znovu. Timeout se exponenciálně zvětšuje s počáteční hodnotou 500 ms. Pakety se tedy odesílají s intervalem $\{0{,}5;\ 1;\ 2;\ 4;\ \ldots\}$ sekundy. Pokud ani po odeslání zprávy s timeoutem 32 sekund nepřijde žádná odpověď, je veškerá komunikace (dialog) jednostraně ukončena.

Protokol SDP

SDP – Session Description Protocol definovaný v RFC 4566 [12] je protokol sloužící k vyjednání parametrů multimediálních streamů (session). Při VoIP hovorech sestavovaných pomocí proto-

[8] Zde je míněno zabezpečení komunikace proti výpadkům, duplicitám a prohození paketů, ne kryptografické zabezpečení.

kolu SIP tvoří tělo SIP zpráv, které vedou k sestavení session. Specifikace dovoluje pomocí SDP sestavit nejenom VoIP hovor, ale prakticky libovolný (nejen) multimediální stream. Následující krátký popis je nicméně zaměřen pouze na nejzákladnější povinné parametry nutné k sestavení session VoIP hovoru.

SDP zpráva je textová a skládá se z množiny záznamů, kde jedna řádka zprávy odpovídá vždy jednomu záznamu. Formát jednotlivých záznamů je obecně `<typ>=<hodnota>`, kde `typ` je vždy určen jediným znakem. SDP definuje tyto povinné záznamy v tomto pořadí:

Protocol version (v) – udává verzi protokolu. Vsoučasné době vždy 0.

Owner/creator (o) – udává zdroj (originator) session. Kromě jména vlastníka session udává i jedinečný identifikátor session a síťovou adresu zdroje.

Session name (s) – udává jméno session.

Connection data (c) – udává adresu pro připojení k danému multimediálnímu streamu. Informace v části specifické pro daný stream „přepisují“ informace z globální části. (viz dále)

Time (t) – udává časové informace o streamu (začátek, konec). Pokud jsou oba údaje rovny 0, jedná se o permanentní spojení.

Dále pak zpráva může obsahovat libovolný počet popisů nabízených multimediálních streamů:

Media description (m) – obsahuje informace o typu streamu (audio, video, data), portu na kterém je stream dostupný, informace o transportním protokolu (pro RTP je to `RTP/AVP`) a seznam formátů media (kodeků). Pro RTP spojení se jedná o typ obsahu definovaný v RTP specifikaci. (0 = G.711 PCMU, 8 = G.711 PCMA, ...)

Pokud za `media description` záznamem následují další záznamy, vážou se (až do dalšího `media description` záznamu nebo konce zprávy) k výše uvedenému multimediálnímu streamu. SDP zpráva tak například může obsahovat různou položku `connection data` pro různý multimediální stream.

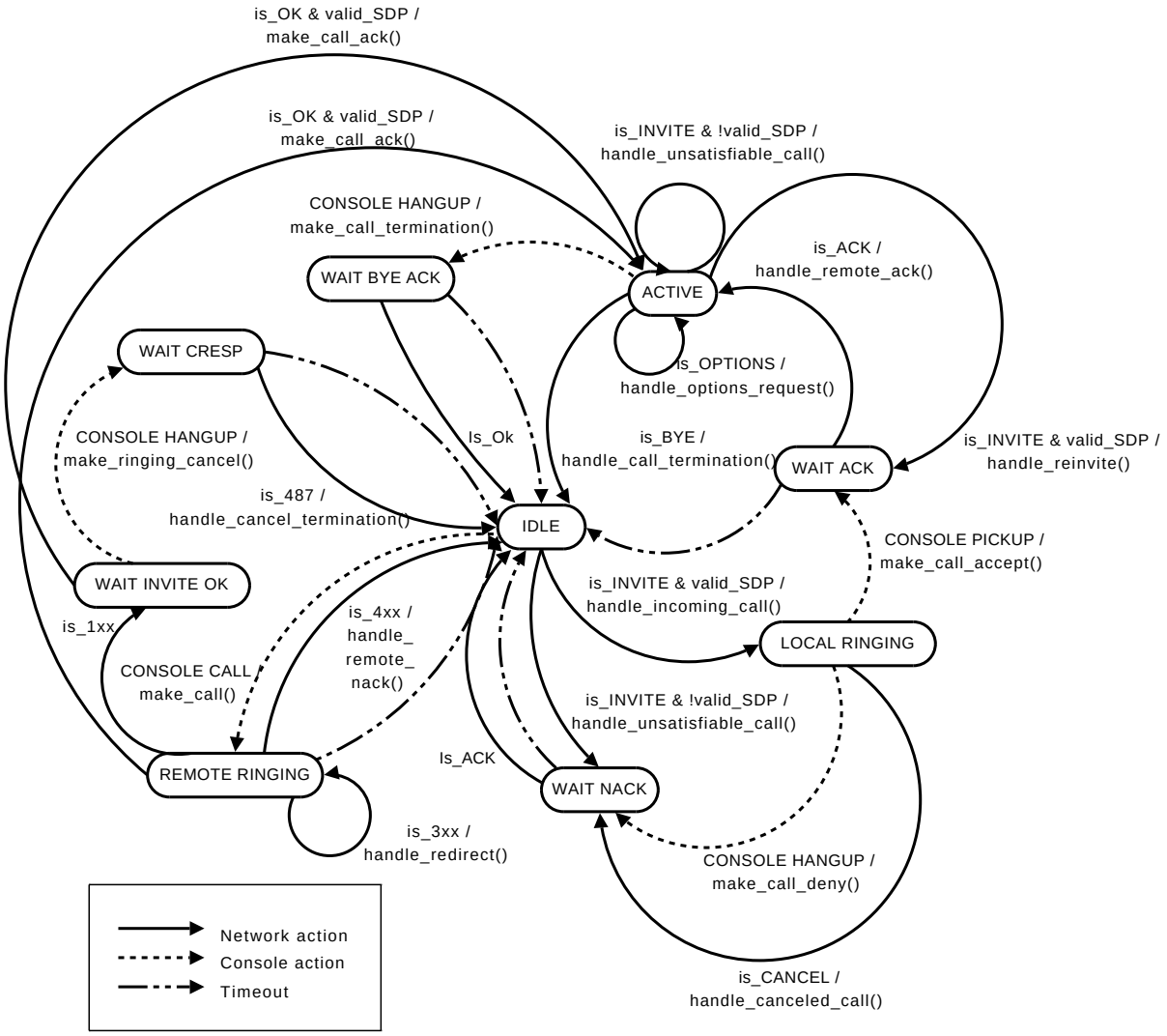
Nejjednodušší SDP zpráva nabízející audio stream ve dvou možných formátech – G.711 PCMA a G.711 PCMU je součástí SIP paketu na obrázku 3.10.

Implementace

Implementace SIP/SDP stacku se skládá ze dvou stěžejních částí – z knihovny `sip.h` poskytující funkce pro práci s SIP/SDP zprávami (parsování/sestavování) a konečného automatu (`sip_fsm.h`) řídícího průběh SIP spojení (dialogu).

Knihovna `sip.h` se skládá ze dvou základních modulů – SIP/SDP parseru (`parser.h`), který parsuje přijaté textové zprávy do definovaných struktur, a funkcí pro sestavování SIP zpráv (`sip.c`). Hlavičkový soubor `sip.h` kromě definic datových struktur pro zprávy obsahuje i makra zjednodušující práci s těmito strukturami. Implementovaný parser je typu LL(1) a je řešen rekurzivním sestupem. Jeho výstupem je buď přijatá zpráva ve formě definované struktury, nebo informace o chybě. Kromě parsování SIP/SDP zpráv slouží parser i ke kontrole syntaxe zadávaných SIP adres.

Parser „rozumí“ pouze určité, klienty při dvoubodových spojeních běžně používané, podmnožině jazyka protokolu. Neimplementuje například podporu pro některé vícenásobné hlavičky, které se mohou vyskytnout při konferenčních hovorech, i podporu pro některé možné, ale v klientech běžně nepoužívané, formáty zápisu některých informací ve zprávě. Výsledkem



Obrázek 3.12: Řídící automat SIP stacku.

příjmu zprávy obsahující dle standardu platný zápis, který ale přesahuje možnosti parseru není žádná odpověď klienta – nedojde tedy k žádnému spojení.[9] Veškeré zprávy generované zařízením zcela odpovídají specifikaci. Gramatiky implementovaných podmnožin jazyka zpráv protokolu SIP i SDP jsou uvedeny v přílohách.

Řídící konečný automat je implementován v souboru `sip_fsm.c`. Jedná se o centrální SW část celého zařízení. Navržený automat není spojením implementací jednotlivých transakčních automatů tak, jak jsou uvedeny v sekci 17 příslušného RFC [14], odpovídá nicméně základní sémantice transakcí. Vstupy a výstupy jsou tři – kromě SIP zpráv a uživatelské konzole se jedná o časovač, který slouží k hlídání časových limitů (timeoutů) odpovědí u síťové komunikace. Grafické znázornění automatu je na obrázku 3.12. Startovacím stavem je stav `IDLE`.

Následující seznam podává stručné informace k jednotlivým stavům a jim náležejícím přecho-

[9] Obecně je sice možné, že takováto zpráva dorazí během dialogu, ale není to příliš pravděpodobné, neboť veškeré potenciálně „nebezpečné“ údaje zpravidla obsahuje již první zpráva INVITE nebo 200 Ok. Pokud by k takové situaci přeci jenom došlo, bude se pro protistranu chování zařízení jevit jako ztráta spojení a dialog ukončí, čímž dojde (důsledkem timeoutu) k ukončení dialogu i na samotném zařízení. V žádném případě tedy nedojde k nějakému nedefinovanému chování zařízení či dokonce k jeho „pádu“.

dům/výstupům a uvádí je do kontextu průběhu SIP dialogu.

IDLE – výchozí stav automatu. V tomto stavu systém přijímá jak SIP zprávy s žádostmi o sestavení hovoru (příchozí hovory), tak žádosti o sestavení hovoru vyvolané uživatelem (odchozí hovory). Je-li příchozí hovor přijatelný (nabízí podporovaný multimediální stream), odešle systém protistraně informaci o vyzvánění a automat přechází do stavu `LOCAL RINGING`. V opačném případě je hovor odmítnut. V případě žádosti o uskutečnění odchozího hovoru odesílá systém volanému zprávu INVITE a přechází do stavu `REMOTE RINGING`.

LOCAL RINGING – lokální vyzvánění. Pokud uživatel hovor přijme, je volajícímu zaslána zpráva `200 Ok` a automat přechází do stavu čekání na potvrzení sestavení session – `WAIT ACK`. Pokud uživatel hovor odmítne, je zaslána zpráva `403 Forbidden` a automat přechází do stavu `WAIT NACK`. Dále v tomto stavu reaguje automat na stornování volání protistranou (požadavek CANCEL).

REMOTE RINGING – čekání na odezvu protistrany. Při příchodu některého z provizorních stavových kódů přechází automat do stavu `WAIT INVITE OK`, pokud je hovor rovnou přijat kódem `200 Ok`, a hovor je přijatelný, spustí se multimediální stream a automat přejde do stavu `ACTIVE`. Pokud nepřijde od protistrany žádná odpověď, je pokus o spojení ukončen nastalým timeoutem. Pokud je odpověď přesměrování – 3xx, je přesměrování potvrzeno ACK zprávou a je inicializováno nové spojení s adrsou získanou z 3xx zprávy. Při chybové odpovědi 4xx případně 5xx, 6xx je odpověď potvrzena (ACK) a dialog ukončen.

WAIT INVITE OK – je prakticky kopií stavu `REMOTE RINGING` s tím rozdílem, že po obdržení provizorního kódu může uživatel ukončit pokus o spojení zasláním `CANCEL` zprávy.

WAIT ACK – je stav, ve kterém automat čeká na potvrzení dohodnuté session protistranou. Po příjmu potvrzení (`ACK`) je spuštěn multimediální stream a automat přechází do stavu `ACTIVE`.

WAIT NACK – je stav, ve kterém SIP stack čeká na potvrzení odmítnutí spojení.

ACTIVE – stav, ve kterém probíhá vlastní multimediální přenos mezi SIP klienty. Pokud je obdržena zpráva `BYE`, je multimediální přenos ukončen, odesláno potvrzení ukončení (`200 Ok`) a automat přechází do stavu `IDLE`. Pokud se rozhodne hovor ukončit lokální uživatel, je rovněž ukončen multimediální stream, protistraně je zaslána žádost `BYE` a automat přechází do stavu `WAIT BYE ACK`. Ve stavu `IDLE` kromě obou žádostí o ukončení hovoru může nastat situace, že vzdálená strana zažádá o změnu parametrů streamu (obvykle se jedná o přesměrování, které provádí „volající“ SIP server, viz obr. 3.11 d)). Pokud lze změnu provést, je stream ukončen, zaslána zpráva o přijmutí změny (`200 Ok`) a automat přechází do stavu `WAIT ACK`, kde je po potvrzení změny stream znovu spuštěn s novými parametry. Nelze-li změnu streamu akceptovat, je odmítnuta příslušným chybovým kódem. Původní dialog však zůstává aktivní. A konečně poslední akcí, která může ve stavu `ACTIVE` nastat, je příchozí požadavek `OPTIONS`. Ten zasílají někteří klienti periodicky během session, aby ověřili zda je protistrana stále dostupná. Odpovědí na `OPTIONS` požadavek je `200 Ok` s SDP.

WAIT BYE ACK – je stav, ve kterém automat čeká na potvrzení ukončení dialogu protistranou.

WAIT CRESP – v tomto stavu je očekáváno potvrzení stornování odchozího volání protistranou.

Na obrázku znázorňujícím automat nejsou z důvodu přehlednosti některé přechody/výstupy zakresleny. V každém stavu kromě IDLE je navíc „smyčka“ sloužící k odmítnutí nově příchozích hovorů stavovým kódem `486 Busy Here`. V každém stavu jsou pak „smyčky“ reagující na duplicitní zprávu odesláním posledně odeslané zprávy a na zprávy jiné než očekávané či INVITE chybovou odpovědí `481 Call/Transaction Does Not Exist`.

3.2.2 RTP stack

RTP stack je část SW zařízení, která se stará o přenos dat hovoru, tedy o obousměrný plně duplexní přenos zvuku. V následující kapitole je popsán samotný systém/formát přenosu – protokol RTP a jeho implementace v zařízení. Text neobsahuje popis přidruženého protokolu RTCP, který zařízení neimplementuje, neboť pro dvoubodovou komunikaci s fixním datovým tokem není potřeba[10] ani rozšiřujících profilů.

Popis protokolu

Real-time Transport Protocol (RTP), poprvé publikovaný IETF v roce 1996 v RFC 1889, je protokol pro přenos multimediálních dat v reálném čase po paketových sítích. Protokol zajišťuje identifikaci typu obsahu streamu, číslování sekvencí dat (jednotlivých RTP paketů), časové značky sekvencí a pomocí přidruženého protokolu RTCP (RTP Control Protocol) definovaném v témže RFC sledování kvality spojení. Kvalitu spojení ovšem žádným způsobem negarantuje, ani nevyhrazuje žádné zdroje pro přenos dat.

Protokol RTP může být obecně využit pro přenos dat skrze libovolnou nižší síťovou vrstvu, která umožňuje demultiplexing datových (RTP) a kontrolních (RTCP) paketů, prakticky se však téměř výhradně pro přenos používá protokol UDP. Specifikace RTP neudává žádný konkrétní UDP port služby, VoIP klienti nicméně pro RTP nejčastěji používají porty 8000 a 9000. RTP nobsahuje žádné mechanismy pro sestavení vlastního spojení, tedy určení adres, portů či typu dat (kodeků) účastnících se zařízení, ty musí být určeny jiným protokolem před zahájením přenosu. RTP zajišťuje pouze přenos multimediálních dat, jinými slovy určuje jejich formát. RTP pakety se skládají z vlastních dat multimediálního streamu a z hlavičky identifikující spojení a pozici aktuální sekvence přenášených dat ve streamu, a to jak časově (timestamp), tak logicky (sekvenční číslo).

RTP pakety mají binární formát s následující strukturou:

Version – udává verzi protokolu. Verze protokolu definovaná v aktuální specifikaci protokolu (RFC 3550 [15]) je 2.

Padding – pokud je tento bit nastaven na „1“, obsahují data několik „vyplňujících“ bajtů, které mají být ignorovány. Počet těchto bajtů udává poslední bajt zprávy.

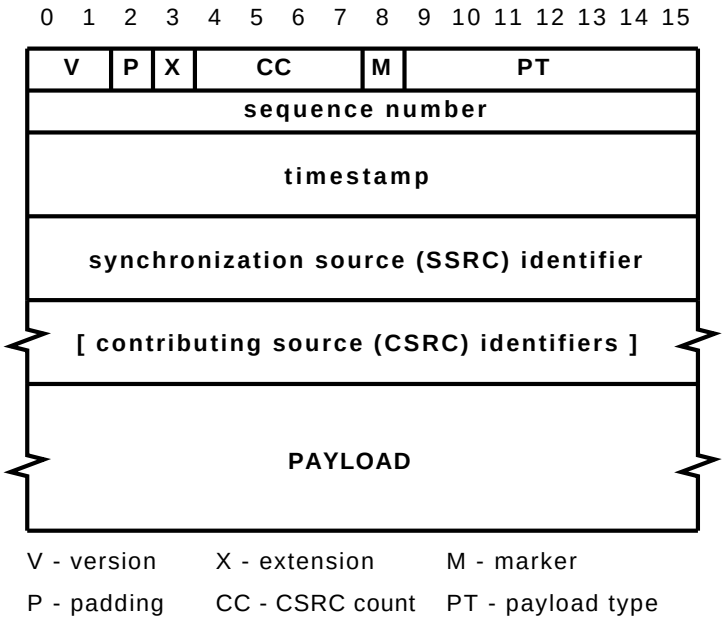
Extension – pokud je bit extension nastaven na „1“, následuje za standardní hlavičkou ještě hlavička rozšiřující.

CSRC count – udává počet záznamů CSRC v hlavičce

marker – význam marker bitu je specifický pro jednotlivé profily určené rozšiřující hlavičkou.

Payload type – sedmibitový identifikátor typu přenášených dat (kodeku)

[10] Modifikace datového toku na základě aktuálního stavu sítě není možná a ostatní funkce RTCP se týkají především multicast streamů pro konferenční spojení.



Obrázek 3.13: Struktura RTP paketu.

Sequence number – sekvenční číslo paketu. S každým dalším paketem se o 1 (modulo 2^{16}) zvyšuje, počáteční hodnota je zvolena náhodně.

Timestamp – časová značka aktuálního úseku dat. Způsob inkrementace tohoto pole je určen typem přenášených dat. V případě zvukového audio streamu s fixní vzorkovací frekvencí (G.711 μlaw, G.711 Alaw) se obvykle čítač zařízení inkrementuje s každým zvukovým vzorkem, pole je tedy inkrementováno o počet přenášených vzorků v paketu.

SSRC – pole SSRC určuje synchronizační zdroj daného streamu, tedy zařízení, které data vysílá. Při point-to-point spojeních je ekvivalentem identifikátoru streamu, v konferenčních hovorech určuje kterému zdroji (účastníkovi konference) náleží aktuální data. Ostatní zdroje dat jsou pak uvedeny v polích `CSRC`.

CSRC – při konferenčním hovoru obsahuje hlavička paketu pro každý aktuálně neaktivní zdroj dat v konferenci jeho identifikátor – `CSRC`.

Struktura paketu je znázorněna na obrázku 3.13.

Implementace

Implementace RTP stacku lze logicky rozdělit několika způsoby na několik částí. První dělení je na část obstarávající odchozí stream a na část obstarávající stream příchozí. Oba dva streamy jsou na sobě zcela nezávislé, jejich zpracování proto provádí taktéž dvě na sobě zcela nezávislá vlákna – `rtp_send_thrad` a `rtp_recv_thread`. V každém z těchto vláken je pak třeba vyřešit dvě podúlohy – příjem/vysílání paketů a záznam/přehrání zvukových dat obsažených v paketech.

Stejně jako u implementace SIP/SDP stacku, je implementace rozdělena na základní funkce pro práci s RTP pakety (`rtp.h`) a na vlastní logiku stacku (`rtp_fsm.h`).

Princip činnosti stacku je založen na práci s výstupní a vstupní frontou paketů. Vstupní fronta je plněna vláknem `rtp_recv_thread` a vyprazdňována obslužnou rutinou přerušení pro

přehrávání zvukového kodeku. Výstupní fronta je pak plněna obslužnou rutinou přerušení pro záznam zvukového kodeku a vyprazdňována vláknem `rtp_send_thread`. Časování odchozích paketů je tak řízeno přes přerušení přímo vzorkovací frekvencí zvukového kodeku. Odchozí pakety jsou z fronty odebírány a odesílány okamžitě, jak je celý paket k dispozici. Pro oba dva podporované kodeky G.711 to znamená vždy po 160 vzorcích. Trochu odlišná situace je na straně příjmu. Zde je implementován buffer, vyrovnávající kolísání velikosti zpoždění paketů – jitter. Proto je při zahájení přenosu dat (stramu) nejdříve vstupní fronta do určité velikosti naplněna a až poté je povoleno její vyprazdňování v obslužné rutině přerušení. Výchozí velikost jitter bufferu je nastavena na 4 pakety, tedy 80 ms (bude diskutováno v závěru).

Při příjmu prvního paketu z nového streamu je zkontrolováno, zda stream odpovídá parametrům dohodnutým v předcházejícím SIP/SDP dialogu (adresa, port, typ dat), při příjmu dalších paketů už je kontrolováno pouze, zda se jedná o RTP paket, SSRC a sekvenční číslo. Je-li obdržen paket s nižším sekvenčím číslem než je aktuální očekávané číslo, je zahozen. Je-li obdržen paket s vyšším sekvenčním číslem, je předpokládána ztráta paketu a o jedna inkrementované sekvenční číslo se stává očekávaným sekvenčním číslem. Pakety tedy nejsou nijak řazeny, pokud dojdou mimo pořadí, jsou pakety s nižším sekvenčním číslem jednoduše zahozeny.

Za „běhu“ programu může teoreticky nastat několik situací, kdy by mohlo dojít k souběhu (race condition), proto je třeba místa, kde by k tomu mohlo dojít, patřičně ošetřit. Ošetření race conditions při přidávání/odebírání z fronty paketů je řešeno pomocí zakázání přerušení v kritických sekcích vláken, neboť tuto situaci nelze řešit synchronizačními prostředky OS. Důvodem je fakt, že odebírání/přidávání z/do fronty probíhá v obslužných rutinách přerušení, které jsou mimo „vliv“ OS.

3.2.3 DNS resolver

Jelikož knihovna lwip, obstarávající v zařízení TCP/IP stack, neposkytuje podporu pro překlad IP adres na doménová jména a žádná vhodná, volně dostupná a dostatečně na zdroje nenáročná knihovna nebyla nalezena, obsahuje systém vlastní implementaci DNS resolveru. Knihovna implementující DNS resolver, nacházející se v souborech `resolver.h` a `resolver.c` implementuje jedinou funkci – `gethostbyname()`, která je kompatibilní se stejnojmennou POSIX/BSD implementací. Oproti POSIX normě má vlastní implementace nicméně některá zjednodušení vyplývající z předpokládaného využití – funkce vrací pouze první A DNS záznam pro danou doménu, nikoliv seznam všech IP adres náležejících k dané doméně a především, knihovna umí pracovat pouze s rekurzivními DNS servery.

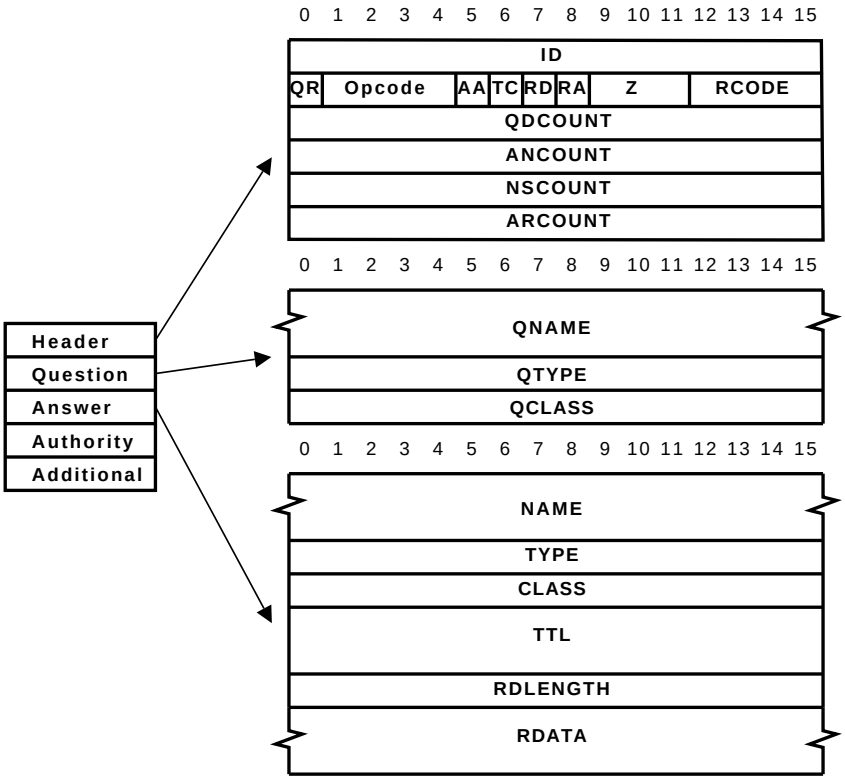
Následující sekce krátce popisují protokol DNS z pohledu resolveru a jeho komunikaci s DNS serverem (nejedná se v žádném případě o kompletní a detailní popis celého systému!), implementaci a použití implementovaného resolveru.

Popis protokolu

Systém a protokol doménových jmen – DNS (Domain Name System) je definován v RFC 1034 a 1035 z roku 1987. Tyto RFC popisují hierarchickou strukturu DNS serverů sloužících (nejenom) k překladu doménových jmen na IP adresy a protokoly pro výměnu zpráv, dotazů a odpovědí, mezi servery a klienty (resolvery).

DNS kromě překladu doménových jmen na IP adresy (A záznam) umožňuje i překlad opačný (PTR záznam), v DNS záznamech jsou dále uloženy i některé další informace o doméně, například jméno poštovního serveru domény (MX záznam) či jméno autoritativního nameserveru domény (NS záznam). Každý paket proto obsahuje informace o typu dotazu/odpověďi. Resolver však běžně pracuje pouze s A záznamy.

Velice zjednodušeně lze říci, že se v internetu vyskytují dva druhy DNS serverů. Nerekurzivní, spravující záznamy pro jednotlivé domény a rekurzivní, které jsou využívány v lokálních



Obrázek 3.14: Struktura DNS paketu.

sítích nebo sítích ISP pro překlad doménových jmen jednotlivými klienty. Zatímco nerekurzivní server odpoví klientovi pouze na dotazy na „své vlastní“ domény či adresy, rekurzivní převezme za klienta vyhledání správného DNS serveru, kterého je třeba se zeptat, a vrátí mu odpověď na dotaz na jakoukoliv doménu či adresu.

Jelikož běžný resolver téměř vždy pracuje pouze s jedním, konfigurací systému daným, rekurzivním DNS serverem, je princip jeho funkce velice jednoduchý. Při požadavku na překlad zašle dotaz „svému“ DNS serveru a čeká na jeho odpověď. Z důvodu urychlení překladu a snížení nároků na síťové připojení většina resolverů navíc obsahuje lokální cache a dotaz na DNS server zasílá pouze v případě, že v této cache požadovaný záznam nenalezne.

Komunikace mezi resolverem a serverem standardně probíhá pomocí protokolu UDP na portu 53. DNS pakety pro dotazy a odpovědi mají formálně stejný formát, pomocí příslušných polí v hlavičce paketu se nicméně určuje jaké a kolik částí kromě povinné hlavičky paket obsahuje. DNS dotazy se tak obvykle skládají pouze z hlavičky a dotazu, zatímco odpovědi obsahují navíc i RR (Resource Record) s odpovědí. Struktura DNS paketu je zobrazena na obrázku 3.14.

Význam nejdůležitějších[11] polí hlavičky DNS paketu je následující:

ID – 16-bitový identifikátor umožňující rozlišit jednotlivé dotazy/odpovědi. Odpověď příslušející k danému dotaz má stejné `ID` jako dotaz samotný.

QR – bit určující, zda je daná zpráva dotaz („0“) nebo odpověď („1“).

OPCODE – určuje typ dotazu. Pro standardní dotazy má hodnotu 0.

[11] Pole, se kterými pracuje implementace.

RD – bit určující, zda se jedná o rekurzivní dotaz.

RCODE – kód odpovědi. Toto pole má nenulovou hodnotu, pokud při zpracování dotazu serverem došlo k chybě.

QDCOUNT – udává počet dotazů v paketu.

ANCOUNT – udává počet RR odpovědi.

Část DNS paketu obsahující dotaz se skládá ze tří polí:

QNAME – dotazované doménové jméno.

QTYPE – typ dotazu (A, MX, NS, ...).

QCLASS – třída dotazu (prakticky výhradně IN – Internet).

Položky RR obsahujícího odpověď pak mají tento význam:

NAME – dotazované doménové jméno.

TYPE – typ dat v položce `RDATA`.

CLASS – třída dat v položce `RDATA`.

TTL – doba platnosti odpovědi.

RDLENGTH – délka položky `RDATA` v bytech

RDATA – data odpovědi. Formát položky je závislý na typu a třídě RR, pro A dotaz se jedná o 4 byty IP adresy.

Položky `QNAME` a `NAME` udávající doménové jméno nejsou prosté ASCII řetězce, ale mají speciální formát – místo tečky oddělující jednotlivé řády domény je vždy byte udávající délku následující části, v terminologii DNS nazývané label. Řetězec je vždy ukončen znakem s ASCII kódem 0.

Systém navíc zavádí tzv. kompresi řetězců. Pokud mají nejvyšší 2 bity v bytu udávajícím délku následující části hodnotu „1“, nevyjadřuje byte délku následujícího labelu, ale jedná se o 2B ukazatel, který udává offset od začátku paketu, kde jméno domény pokračuje. Podrobnosti viz RFC 1035 [13].

Implementace

Algoritmus resolveru je velice jednoduchý, k výše popsanému schématu komunikace dotaz-odpověď přidává pouze práci s integrovanou cache:

1. Zkontroluj přítomnosti dotazovaného záznamu v cache. Pokud je záznam přítomen a je stále platný, vrať tento záznam a skonči.
2. Vytvoř DNS dotaz a odešli jej na DNS server.
3. Čekej na odpověď. Pokud odpověď nedorazí v definovaném čase, vrať chybu (NULL) a skonči.
4. Rozparsuj přijatý paket. Pokud obsahuje platná data, přidej záznam do cache na místo některého z neplatných záznamů. Pokud jsou všechny záznamy v cache platné, přepiš ten s nejnižší zbývající dobou platnosti[12]. V případě, že odpověďí serveru je chybová zpráva (`RCODE != 0`), vrať NULL.

Resolver je kromě systémových knihoven závislý na knihovně časovače `timer.h` a knihovně síťových funkcí `network.h`.

[12] Lze očekávat, že takový záznam byl přidán jako první a tudíž je nejnižší pravděpodobnost, že bude v nejbližší době opět potřeba.

Programátorský model

Programátorské rozhraní resolveru je kompatibilní se standardní POSIX/BSD verzí funkce `gethostbyname()` z hlavičkového souboru `netdb.h`. V hlavičkovém souboru vlastního resolveru – `resolver.h` jsou kromě této funkce definovány také funkce pro inicializaci resolveru a nastavení DNS serveru.

SYNOPSE

```
struct hostent {
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_length;
    char **h_addr_list;
#define h_addr h_addr_list[0]
};

struct hostent *gethostbyname(char *name);
void resolver_init();
void resolver_set_server(net_in_addr_t server);
```

POPIS

Funkce `gethostbyname()` vrací ukazatel na strukturu `hostent` obsahující dotazované doménové jméno (`h_name`) a jemu náležející IP adresu (`h_addr`). `h_aliases` je vždy NULL (resolver nepodporuje aliasy), `h_addrtype` AF_INET a `h_length` 4. V případě neúspěchu vrací funkce NULL. Funkce není současně volatelná z více vláken (reentrant) – každé volání přepíše hodnotu v globální datové oblasti struktury `hostent`, kterou funkce používá.

K inicializaci resolveru slouží funkce `resolver_init()`, která musí být zavolána před prvním použitím funkce `gethostbyname()`.

Funkce `resolver_set_server()` nastavuje IP adresu DNS serveru, který resolver používá. Server musí být nastaven před prvním voláním `gethostbyname()` a může být měněna v průběhu programu.

3.2.4 lwIP DHCP patch

Následující text popisuje postup zprovoznění DHCP klienta v knihovně lwip, který ve verzi (portu) knihovny obsažené v EDK 9.1 oficiálně není podporován. Knihovna lwip však „obecně“ podporu DHCP obsahuje.

Součástí zdrojových kódů knihovny je i soubor `dhcp.c`, kde je prakticky kompletní DHCP klient implementován. K jeho zprovoznění je třeba patřičným způsobem upravit makefile (`Makefile_mb` pro MicroBlaze HW platformu a/nebo `Makefile_ppc` pro PowerPC HW platformu) knihovny a ve vlastní aplikaci vyřešit periodické volání funkcí `dhcp_fine_tmr()` a `dhcp_coarse_tmr()` v patřičných intervalech. DHCP klient také standardně neumí předávat systému informace o adrese DNS serveru získané[13] pomocí DHCP, ke zprovoznění této funkce je třeba upravit zdrojový kód knihovny lwip, konkrétně soubor `dhcp.c` a hlavičkové soubory `netif.h` a `dhcp.h`. Na přiloženém CD jsou tyto upravené soubory, stejně tak jako upravený makefile, k nalezení v adresáři `lwip_patch`.

K aplikování „patchů“ je třeba přepsat těmito soubory soubory z EDK, které se nachází v adresáři `$EDK/sw/ThirdParty/sw_services/lwip_v2_00_a/src`, kde `$EDK` značí adresář,

[13] V DHCP dotazu o adresu DNS serveru ani nežádá.

kde je nainstalováno samotné EDK. Soubory makefile se nacházejí přímo v tomto adresáři. Soubor `dhcp.c` je v podadresáři `/lwip/src/core`, soubory `netif.h` a `dhcp.h` pak v podadresáři `/lwip/src/include/lwip`.

V aplikaci, která má DHCP klienta využívat, je třeba do úspěšného přidělení IP adresy pravidelně s periodou `DHCP_FINE_TIMER_MSECS` milisekund volat funkci `dhcp_fine_tmr()`. Pokud má DHCP klient být i po přidělení IP adresy dále aktivní a kontrolovat možnou změnu IP adresy zaslanou DHCP serverem, je navíc třeba každých `DHCP_COARSE_TIMER_SECS` sekund volat funkci `dhcp_coarse_tmr()`. Provedené úpravy ve zdrojových kódech lwip knihovny navíc v případě použití DHCP klienta rozšiřují strukturu `netif` obsahující informace o síťovém rozhraní (IP adresa, maska sítě, výchozí brána) o prvek `dns` (stejně jako u ostatních jmenovaných parametrů IP se jedná o datový typ `struct ip_addr`), který obsahuje adresu DNS serveru přidělenou DHCP serverem.

Kompletní příklad kódu aplikace, je uveden v anglicky psané dokumentaci k patchům v příloze *Xilinx lwip DHCP mini HOWTO*.

3.3 Uživatelská konzole

Kapitola *Uživatelská konzole* popisuje moduly/knihovny realizující logiku ovládání zařízení. Jedná se o knihovny pro práci s LCD displejem a PS/2 klávesnicí a jejich propojení.

Klávesnice

Pro práci s klávesnicí slouží v systému knihovna `keyboard.h`. Jejím primárním účelem je sekvenci přijatých bajtů (scancodes) z klávesnice převést na ASCII kód, který je možné poslat na displej. Komunikační protokol klávesnice totiž rozlišuje zmáčknutí klávesy (make code) a její uvolnění (break code). Pokud scancode vyjadřuje uvolnění klávesy, předchází mu vždy znak s kódem `0xF0`. Některým „rozšířeným“ znakům, například funkčním klávesám F1-F10 pak navíc předchází další znak – `0xE0`.

Knihovna pro práci s klávesnicí obsahuje kromě funkcí na inicializaci klávesnice a knihovny samotné pouze jedinou funkci:

```
char kbd_getchar();
```

Ta je obdobou funkce `getchar()` ze standardní C I/O knihovny `stdio`. Na rozdíl od této však pracuje v neblokujícím režimu – pokud není žádný nový znak dostupný, vrací 0. Kromě ASCII kódů znaků odpovídajících zmáčknutým klávesám vrací funkce ještě několik speciálních kódů - `0x01` pro Enter, `0x02` pro ESC a `0x03` pro backspace.

Knihovna nepodporuje všechny klávesy standardní PC104 klávesnice, ale jen jejich omezenou množinu potřebnou pro ovládání zařízení. Podporovány jsou alfanumerické a speciální znaky anglické klávesnice (včetně jejich „velkých“ variant, knihovna tedy podporuje klávesu shift), z funkčních kláves pak Enter, ESC a backspace.

LCD

Knihovna pro práci s LCD displejem – `lcd.h`, obstarává uživatelský výstup informací o stavu zařízení. Formát výstupu je navržen tak, že první řádek dvouřádkového LCD vždy obsahuje informaci o stavu zařízení (například "Ready" – připraveno k volání/příjmu hovoru, "Incoming call" – příchozí hovor, "Call active" – aktivní hovor, ...) a druhá řádka buď kurzor a text vstupu z klávesnice, nebo doplňující informace o stavu. Pokud je konzole ve stavu, kdy je jedinou možnou akcí potvrzení nebo odmítnutí pomocí Enter/ESC, je kurzor deaktivován. Pokud je text, který je třeba zobrazit na jedné řádce, delší než maximální počet znaků na řádce

(16), je text buď automaticky, nebo v závislosti na vstupu uživatele posouván. Tomuto systému zobrazení pak odpovídá množina funkcí v knihovně:

```
void lcd_prints(char *string);
void lcd_prints2(char *string);
void lcd_putchar(char c);
void lcd_disable_cursor();
void lcd_enable_cursor();
void lcd_shift_left();
void lcd_shift_right();
void lcd_backspace();
```

lcd_prints2() – funkce vypíše řetězec `string` na aktuální pozici kurzoru[14] na LCD.

lcd_prints() – smaže obsah LCD, vypíše řetězec `string` od prvního znaku prvního řádku displeje a následně nastaví kurzor na začátek druhé řádky.

lcd_putchar() – vypíše znak `c` na aktuální pozici kurzoru.

lcd_*_cursor() – `lcd_enable_cursor` a `lcd_disable_cursor` zapínají/vypínají zobrazování kurzoru.

lcd_shift_*() – funkce pro posun obsahu displeje o jednu pozici směrem do prava (`lcd_shift_right()`) a do leva (`lcd_shift_left()`).

lcd_backspace() – smaže znak na aktuální pozici kurzoru a posune kurzor o jeden znak do leva.

Funkce pracují s „rozšířenou“ pamětí (znaky 16-39) interního řadiče LCD, tedy se znaky, které jsou v paměti interního řadiče, ale nejsou zobrazovány. Velikost řetězců, se kterými mohou funkce pracovat, je tak omezena na 40 znaků.

Řídící automat

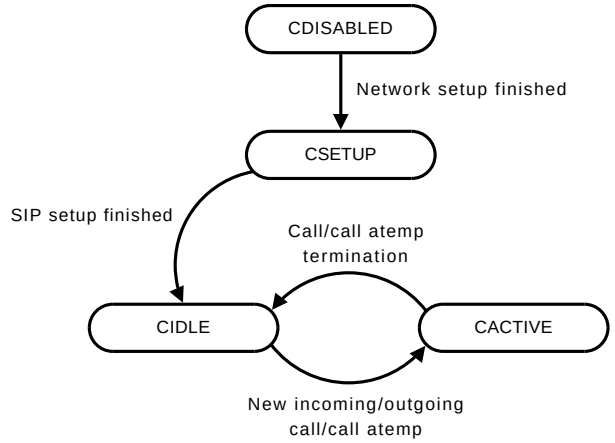
Řídící automat uživatelské konzole (`console.h`) má, nepočítáme-li stavy, ve kterých se konzole nachází během inicializace a konfigurace zařízení, dva stavy. Ve stavu `CIDLE` je zařízení „nečinné“, je povolen uživatelský vstup a po zadání platné SIP adresy je zahájen pokus o spojení a uživatelská konzole přechází do stavu `CACTIVE`. Do stavu `CACTIVE` přechází konzole i v případě, že zařízení (konečný automat SIP stacku) zaznamená příchozí hovor. Ve stavu `CACTIVE` je potlačen uživatelský vstup, konzole reaguje pouze na zmáčknutí kláves Enter a ESC. Tyto akce jsou předány automatu SIP stacku, který dle svého aktuálního stavu a zmáčknuté klávesy nastaví odpovídající stav konzole, tedy setrvání ve stavu `CACTIVE` (např. přechod od vyzvánění k aktivnímu hovoru), nebo návrat do stavu `CIDLE` (jakékoliv ukončení spojení nebo pokusu o spojení). Zjednodušené schéma řídícího automatu konzole je zobrazeno na obrázku 3.15.

3.4 Propojení modulů a konfigurace systému

Propojení HW/SW modulů odpovídá schématům a popisu z kapitoly 2.3.2. „Hlavní program“, který inicializuje zařízení a spouští jednotlivá vlákna, je krátce popsán v části *Hlavní SW modul*.

Kromě popisu inicializace a propojení jednotlivých SW modulů v hlavním programu kapitola dále popisuje specifickou konfiguraci použitých systémových HW/SW komponent, tedy nutné nastavení, aby systém fungoval jako celek. Jsou zde diskutovány HW limity zařízení a optimální konfigurace SW modulů vzhledem k optimální funkci zařízení.

[14] Přesněji aktuální pozici v paměti znaků – DD RAM – displeje, která může být obecně jiná než pozice na které je aktuálně zobrazován kurzor.



Obrázek 3.15: Řídící automat uživatelské konzole.

Hlavní SW modul

Úkolem „hlavního programu“ nacházejícího se v souboru `main.c` je spuštění OS Xilkernel, inicializace knihovny lwip a síťového rozhraní vůbec a spuštění jednotlivých „výkonných“ vláken (console, sip, rtp_send, rtp_recv). OS Xilkernel je nakonfigurován (viz konfigurace SW níže) tak, aby po jeho spuštění pomocí funkce `xmk()` z `main()` funkce programu byla automaticky spuštěna dvě vlákna. Vlákno konzole – `console_thread` a vlákno sloužící k inicializaci SW komponent pracujících s knihovnou lwip – `socket_thread`. V tomto vlákně je provedena inicializace lwip a je spuštěna konfigurace síťového rozhraní zařízení a to v závislosti na nastavení (nastavuje v hlavním konfiguračním souboru `globals.h`) se kterým byl SW přeložen buď pomocí DHCP, nebo staticky.

Po úspěšné konfiguraci síťového rozhraní přechází systém do stavu, ve kterém je uživateli umožněna konfigurace lokální SIP adresy. Po nastavení SIP adresy jsou pak spuštěna zbývající vlákna SIP a RTP stacků a zařízení je zcela připraveno k provozu.

Konfigurace HW

HW nároky zařízení lze velice hrubě rozdělit na nároky na paměť a nároky na výkon (rychlost). Nároky na paměť RAM jsou probírány níže v sekci SW, neboť velikost potřebné paměti je určena SW částí zařízení, zde budou diskutovány pouze nároky na velikost BRAM, které vzhledem k jejímu výhradnímu[15] použití jako cache CPU spadají v zásadě pod nároky na výkon.

Základní složka výkonu je, stejně jako u každého jiného mikroprocesorového systému tohoto typu, určena rychlostí systémových hodin, tedy frekvencí, s jakou jsou vykonávány jednotlivé instrukce CPU. Při konstrukci zařízení je vždy snahou dosáhnout frekvence co nejnižší, neboť nižší frekvence znamená nižší spotřebu energie a obecně nižší nároky na technologii chipu. Minimální teoretická frekvence, na které může být provozován procesor MicroBlaze v navrženém systému je 66 MHz, což je minimum pro funkčnost použité DDR RAM. V projektu na přiloženém CD je zvolena frekvence 75 MHz, která již 100% zaručuje potřebý výkon pro funkci zařízení, tedy výkon potřebný pro zpracování dat v reálném čase.

Zásadní roli na výkon CPU má dle provedených zevrubných testů použití cache. Bez použití cache[16] výkon CPU rapidně klesá. Například síťová odezva zařízení – round-trip time (ping) se

[15] Nepočítáme-li využití její malé části pro inicializační sekvenci, která „nastartuje“ provádění programu z RAM, tzv. bootloop.

[16] U systému, který má data/kód v RAM. U systému, kde již jsou veškerá data/kód v BRAM cache samozřejmě postrádá smysl.

při jinak stejných podmínkách při deaktivaci cache zvýší na dvojnásobek, z 8 ms na 16 ms! Proto je, jak již bylo uvedeno v kapitole 2.3.2 systém vybaven 8 kB instrukční a 8 kB datové cache, což dohromady tvoří maximální velikost BRAM dostupnou pro cache na desce Spartan-3E starter kit board.

Pozorovatelný nárůst výkonu dále přináší integrovaná násobička a barrel shifter. CPU je naopak nakofigurován bez podpory obvodů v plovoucí řádové čárce (floating point unit) – v celém kódu aplikace není použita jediná neceločíselná proměnná a nejinak tomu pravděpodobně bude i v systémových knihovnách.

Celkové HW nároky zařízení na zdroje FPGA při výše uvedené konfiguraci uvádí tabulky 3.4 a 3.5. První z nich uvádí zdroje alokované při syntéze HW pro desku ML403, tedy FPGA chip Virtex-4, typ XC4VFX12-FF668-10C. Druhá tabulka uvádí alokované zdroje při syntéze pro desku Spartan-3E starter kit board[17] s FPGA Spartan-3E, typ XC3S500E-4FG320C.

Resource Type	**Used**	**Percent**
Number of BSCANs	1 out of 4	25
Number of BUFGs	10 out of 32	31
Number of DCM_ADVs	3 out of 4	75
Number of DSP48s	3 out of 32	9
Number of ILOGICs	46 out of 320	14
Number of External IOBs	96 out of 320	30
Number of LOCed IOBs	96 out of 96	100
Number of OLOGICs	67 out of 320	20
Number of RAMB16s	16 out of 36	44
Number of Slices	4193 out of 5472	76
Number of SLICEMs	500 out of 2736	18

Tabulka 3.4: Alokované zdroje FPGA - Virtex 4.

Resource Type	**Used**	**Percent**
Number of External IOBs	75 out of 232	32
Number of LOCed IOBs	75 out of 75	100
Number of BSCANs	1 out of 1	100
Number of BUFGMUXs	10 out of 24	41
Number of DCMs	3 out of 4	75
Number of MULT18X18SIOs	3 out of 20	15
Number of RAMB16s	16 out of 20	80
Number of Slices	3988 out of 4656	85
Number of SLICEMs	493 out of 2328	21

Tabulka 3.5: Alokované zdroje FPGA - Spartan-3E.

Konfigurace SW

Nastavení SW platformy nutné pro správnou funkci zařízení je mnohem komplexnější a složitější záležitost, než nastavení HW platformy. Zapotřebí je správně nastavit OS Xilkernel, knihovnu lwip i rozložení SW částí v paměti dané tzv. linker scriptem.

[17] Deska Spartan-3E Starter board neobsahuje AC97 kodek, při syntéze byly výstupy řadiče AC97 připojeny na piny rozšiřujícího konektoru.

Kromě nutného nastavení Xilkernelu tak, aby odpovídal HW systému, tedy správného určení standardního vstupu/výstupu, řadiče přerušení, systémového časovače a frekvence systémových hodin je třeba pro správnou funkci provést i některá další nastavení. Pro správnou funkci knihovny lwip v socket modu je třeba zapnout v Xilkernelu podporu uživatelských časovačů a semaforů, což je zmíněno i v dokumentaci lwip.

Co zmíněno již není a bylo třeba zjistit experimentálně, jsou hodnoty některých limitů, které pro systém rozsahu, jakým je implementovaná SW část zařízení, mají ve výchozím nastavení příliš malou hodnotu. Předně je třeba zvýšit paměť stacku jednotlivých vláken Xilkernelu. Optimální hodnota pro realizovaný systém byla experimentálně stanovena na 64 kB. Dále je vhodné zvýšit maximální počet operačním systémem podporovaných semaforů, který jsou využívány knihovnou lwip a především je nutné zvýšit frekvenci přepínání kontextu (parametr `systmr_interval`) na co nejmenší hodnotu, výchozí perioda 10 ms již prakticky nedovoluje zpracovávat plně duplexní síťový provoz RTP streamu potřebnou rychlostí. Experimentálně byla optimální hodnota, kdy je přepínání kontextu dostatečně rychlé a zároveň se stihne vykonat dostatečné množství „práce“, určena jako 2 ms.

Konečně poslední změnou oproti standardní konfiguraci je nastavení „statických“ vláken, tedy vláken, která jsou automaticky spuštěna při startu OS. V našem případě se jedná o vlákna `socket_thread` a `console_thread`. Kompletní nastavení Xilkernelu v konfiguraci SW platformy (soubor `system.mss`) pak vypadá následovně:

```
BEGIN OS
 PARAMETER OS_NAME = xilkernel
 PARAMETER OS_VER = 3.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER stdin = RS232_DCE
 PARAMETER stdout = RS232_DCE
 PARAMETER sysintc_spec = opb_intc_0
 PARAMETER systmr_dev = opb_timer_1
 PARAMETER systmr_freq = 75000000
 PARAMETER systmr_interval = 2
 PARAMETER config_time = true
 PARAMETER config_sema = true
 PARAMETER pthread_stack_size = 65536
 PARAMETER max_sem = 32
 PARAMETER max_pthread_mutex = 32
 PARAMETER static_pthread_table = ((socket_thread,1),(console_thread,1))
END
```

Změny v konfiguraci je nutné provést i pro knihovnu lwip. Výchozí typ API knihovny kterým je `RAW_API` je třeba změnit na `SOCKETS_API`. Dále je třeba nastavit výchozí MAC adresu Ethernet adaptéru. Vzhledem k tomu, že aplikace nepracuje s TCP ale pouze s UDP, je podpora pro TCP „vypnuta“. Celý záznam v konfiguračním souboru SW platformy vypadí takto:

```
BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = lwip
 PARAMETER LIBRARY_VER = 2.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER api_mode = SOCKETS_API
 PARAMETER lwip_tcp = false
 PARAMETER emaclite_instances = ((Ethernet_MAC,0x00,0x0A,0x35,0x01,0x02,0x03))
END
```

Poslední věcí, kterou je třeba při konfiguraci SW platformy zařízení provést je vygenerování linker scriptu. Ten určuje rozložení jednotlivých segmentů programu v paměti. Velikosti jednotlivých segmentů nastaví EDK automaticky tak, aby odpovídaly vytvořenému ELF souboru pogramu. „Ručně“ je však třeba nastavit velikost heapu a stacku systému. Obě hodnoty byly experimentálně určeny na 256 kB.

Paměťové nároky aplikace při výše uvedeném nastavení udává tabulka 3.6. Části `text` (instrukce programu) a `data` (inicializovaná data), které dohromady tvoří paměť, která musí být dostupná před spuštěním programu, zabírají ∼120 kB. Část `bss` (neinicializovaná data), do které spadají statické proměnné programu a stack a heap systému pak má velikost ∼2 MB.

text [B]	**data [B]**	**bss [B]**	**total [B]**
111128	7122	2052868	2171118

Tabulka 3.6: Paměťové nároky systému.

3.5 Ovládání zařízení

Ovládání zařízení je velice jednoduché a odpovídá ovládání běžného telefonu, jen místo číselníku je klávesnice a místo telefonního čísla se zadává SIP adresa. Po spuštění zařízení je navíc třeba zadat vlastní SIP adresu. V zásadě k ovládání stačí znát tato jednoduchá pravidla:

- SIP adresa se zadává vždy jako URL, tedy ve tvaru `sip:user@host`, tedy včetně určení protokolu (sip:). Možný je i formát s udáním jména: `"Jmeno" sip:user@host`.
- Spuštění akce, povolení příchozího hovoru, spuštění vytáčení po zadání adresy, ... – zkrátka všechny „kladné“ akce se provádí pomocí klávesy Enter (Return), naopak všechny „záporné“ akce – odmítnutí příchozího hovoru, zrušení aktuálního pokusu o vytáčení, zavěšení, ..., se provádí pomocí klávesy escape (ESC).

Pokud se po zapnutí zařízení neobjeví výzva k zadání lokální SIP adresy (SIP adresy aktuálního uživatele zařízení) ale hláška "Network error!", znamená to, že přidělení IP adresy pomocí DHCP selhalo. Po zadání lokální adresy se na displeji objeví hláška "Ready" a blikající kurzor. Napsáním SIP adresy a zmáčknutím klávesy Enter dojde k pokusu o spojení se zadanou SIP adresou. Ten lze, stejně jako již probíhající hovor ukončit zmáčknutím klávesy ESC.

V případě příchozího hovoru se na LCD objeví informace o příchozím hovoru, v dolním řádku je jméno[18], nebo SIP adresa volajícího. Souběžně se také rozsvítí LED diody na klávesnici, čímž je uživatel na příchozí hovot upozorněn. Hovor lze pomocí klávesy ESC odmítnout, nebo pomocí klávesy Enter přijmout.

[18] Systém je stejný jako u zobrazování emailových adres, v poštovních klientech – není-li součástí URL adresy jméno, zobrazí se samotná adresa.

4 Testy

Vzhledem k rozsahu práce a související velké šíři implementovaných protokolů a HW/SW komponent/modulů nebyla pro žádnou z implementovaných částí zpracována detailní formální validace. U některých částí (SIP stack) navíc již návrh vycházel z pouhé kompatibility s protokolem a nepočítá se 100% dodržením specifikace ve všech detailech. U všech komponent systému nicméně proběhla určitá forma nesystematických testů, která ověřila funkčnost komponenty.

4.1 Testy řadičů periferií

U testů řadičů periferií lze bez nadsázky říci, že jejich nejdůkladnějším testem byl jejich provoz při vývoji zbytku systému, při kterém byly bez jediné chyby v provozu desítky hodin. Jednotlivé řadiče byly samozřejmě testovány také speciálními aplikacemi zaměřenými pouze na funkci té které komponenty. EDK projekt na přiloženém CD pak kromě vlastního SW VoIP zařízení obsahuje i SW projekt `TestApp_Peripherals`, který je komplexním testem HW/ovladačů řadičů periferií vlastního návrhu (LCD, PS/2) i modifikovaného řadiče AC97 a vlastní implementace jeho ovladače.

LCD

Řadič LCD displeje byl prakticky otestován na LCD obsažených na deskách Spartan-3 starter kit board (LCD Sitronix ST7066U) a ML403 (LCD Lumex LCM-S01602DTR/M) při frekvencích 50, 66, 75 a 100 MHz. Při žádné z testovaných konfigurací nebyly zaznamenány jakékoliv problémy s HW/SW řadiče.

PS/2

Řadič PS/2 byl otestován s několika PS/2 klavesnicemi, opět na obou vývojových deskách. Stejně jako u řadiče LCD se nevyskytly žádné problémy. Otestována byla také komunikace s PS/2 myší, která byla pomocí ovladačů řadiče úspěšně nakonfigurována tak, že z ní bylo dále možné číst vysílaná data o pohybu a zmáčknutých tlačítkách. V původním návrhu umožňoval řadič příjem dat z PS/2 zařízení pouze v polling modu, v průběhu vývoje VoIP zařízení se však ukázalo, že i při nejrychlejším možném přepínání kontextu po 1 ms systém nestačí vstup z PS/2 klávesnice zpracovávat[1]. Tento problém byl nicméně zcela vyřešen rozšířením SW/HW řadiče tak, aby mohl pracovat v interrupt modu.

AC97

Řadič AC97 byl úspěšně otestován a provozován na dvou exemplářích desky ML403. Při pokusu o spojení dvou VoIP zařízení na FPGA desce byl projekt zběžně přeložen i pro desku ML401, která byla k dispozici, zvukový modul však na této konkrétní desce nefungoval. Problém nicméně nebyl z časových důvodů dále zkoumán a pro spojení dvou implementací na FPGA, tedy dvou identických zařízení byl úspěšně použit druhý exemplář desky ML403. Možné je, že při rychle prováděné syntéze byly nastaveny nekorektní parametry, nebo že byl dokonce kodek na desce vadný, stejně tak jako že došlo k problémům s časováním designu řadiče.

[1] Nutno dodat, že toto není nijak překvapivé zjištěn, neboť jednoduchým výpočtem z parametrů PS/2 protokolu lze dojít k zjištění, že jednotlivé znaky mohou ze zařízení přicházet v menších intervalech než 1 ms.

4.2 Testy síťových protokolů

Testy síťových protokolů opět probíhaly stylem „ověření funkčnosti oproti co nejširší množině implementací“. Kromě předpokládaných schémat činnosti byly navíc jednotlivé moduly v rámci možností testovány i na některé chybové stavy, jako přerušení spojení, nedostupnost služby či příjem nevalidních zpráv. U samotného přenosu zvuku (RTP multimediální stream) pak byly navíc ještě kontrolovány kvalitativní parametry spojení (jitter, zpoždění, ztrátovost paketů).

DHCP klient

Konfigurace zařízení pomocí protokolu DHCP, včetně nastavení DNS serveru byla úspěšně ověřena v lokální síti s ISC[2] DHCP serverem („standardní“ linuxový DHCP server) a v domácí síti s DHCP serverem routeru Ovislink 1000R. Stejně jako u všech dalších testovaných protokolů byla síťová komunikace analyzována pomocí síťového analyzátoru Wireshark[3] a vyhodnocena jako odpovídající protokolu.

DNS resolver

Funkčnost DNS resolveru byl ověřena při komunikaci s DNS serverem Bind (Bind je stejně jako výše zmíněný DHCP server produktem ISC a je považován za „etalon“ implementace DNS serveru), a DNS serverem implementovaným v routeru Ovislink 1000R. Všechny provedené testy, ať už se jednalo o dotazy na existující či neexistující záznamy opět proběhly úspěšně. Opětovnými dotazy na stejný záznam byla taktéž ověřena správná funkce implementované cache DNS resolveru.

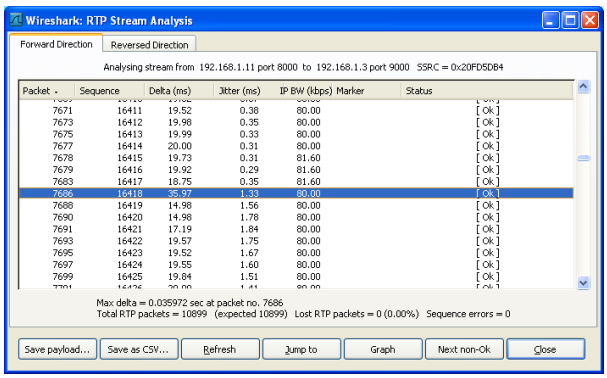
SIP/SDP stack

Nejsložitější testy byly provedeny na ověření funkčnosti SIP stacku, neboť implementovaná podmnožina SIP protokolu je také nejkomplexnějším a nejsložitějším implementovaným protokolem. SIP/SDP zprávy mohou být velice variabilní a je potřeba je, ve srovnání s protokoly s binárním formátem zpráv, komplikovaně parsovat. Taktéž možných situací, které mohou při sestavování hovoru či v jeho průběhu nastat, je celá řada a pro každou z nich je třeba ověřit korektní chování SIP/SDP stacku. SIP/SDP stack musí také umět reagovat na celou řadu chybových stavů, přičemž vždy musí být zajištěno, že se při případné chybě systém automaticky vrátí do výchozího stavu, nebo toho bude schopen na žádost uživatele. Navržený automat (obr. 3.12) tento požadavek formálně splňuje, což lze dokázat pouhým faktem, že z každého stavu vede do výchozího stavu přechod typu ”timeout” nebo ”akce uživatelské konzole”. Kromě formální stránky je nicméně samozřejmě třeba ověřit i samotnou implementaci.

SIP stack byl proto podroben rozsáhlým testům spojení (navázování, ukončování, odmítání, ...) simulujících běžný provoz VoIP telefonu s několika SIP klienty. Jako SIP klient protistrany byly úspěšně vyzkoušeny tyto SW VoIP telefony: Ekiga[4], Twinkle[5], wxCommunicator [6], PJSUA[7] a SIP server Asterisk[8].

Při zkoumaných scénářích nebylo zjištěno žádné chybné či neočekávané chování. SIP stack fungoval dle očekávání jak při peer-to-peer spojeních, tak při komunikaci za přítomnosti SIP serveru Asterisk.

[2] http://www.isc.org/
[3] http://www.wireshark.org/
[4] http://www.gnomemeeting.org/
[5] http://www.twinklephone.com
[6] http://wxcommunicator.sourceforge.net
[7] http://www.pjsip.org
[8] http://www.asterisk.org/



Obrázek 4.1: Analýza RTP streamu generovaného zařízením.

RTP stack

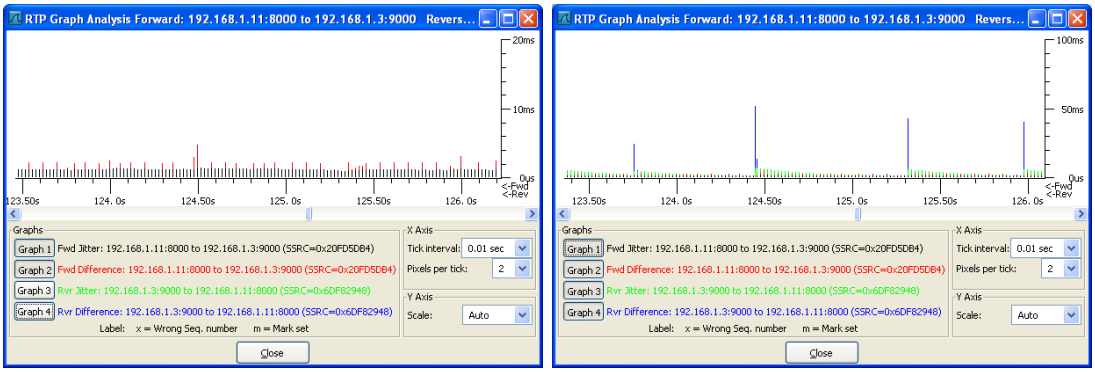
RTP stack byl testován opět pomocí testů spojení s výše uvedeným SIP VoIP software. U obou dvou implementovaných kodeků byl přenos zvuku funkční proti všem testovaným klientům. Bohužel však při RTP spojení dochází k pádům aplikace, jejichž příčiny se přes veškeré úsilí nepodařilo nalézt. Pád může nastat již po pár desítkách vteřin hovoru, ale byly uskutečněny i hovory trvající déle než 30 min. Průměrná doba hovoru, po které dojde k „pádu“ zařízení, je přibližně 10 min. To je také důvod, proč se chyba velmi špatně hledá – nelze ji nijak snadno reprodukovat. S největší pravděpodobností dojde za běhu k nějakému souběhu (race condition), ale ani snahy o zabezpečení všech možných potenciálně nebezpečných míst nevedly k vyřešení problému. Je také velmi pravděpodobné, že chyba je v kódu lwip knihovny a projevuje se jen při intenzivním provozu, který obousměrná RTP komunikace vyvolá.

Jádro knihovny lwip není obecně thread-safe, podporu pro vícevláknové prostředí by měla zajišťovat vrstva socketů, která k tomuto účelů využívá synchronizačních prostředků OS. Bohužel, port této vrstvy na OS Xilkernel pravděpodobně není zcela „dotažen“. V průběhu implementace se vyskytlo několik dalších problémů s touto knihovnou, například „zatuhnutí“ systému při použití funkce `select()` ve více vláknech. Stejný kód přitom bez problémů fungoval na OS linux[9]. V dokumentaci ke Xilinx portu knihovny [6] není vícevláknový provoz knihovny nijak detailně rozebírán, zmiňována je pouze nutnost spouštět všechny vlákna pracující s knihovnou lwip z jednoho vlákna, ve kterém byla knihovna lwip inicializována, což je v případě realizované aplikace dodrženo.

V diskusních fórech na internetu jsou navíc problémy s Xilinx portem knihovny lwip při větší zátěži a/nebo více aktivních socketech poměrně často zmiňovány.

Parametry generovaného streamu byly kromě subjektivního hodnocení kvality přenosu, která se nijak neliší od kvality dosahované při spojení dvou „běžných“ SIP telefonů s G.711 kodekem, také analyzovány pomocí funkcí programu Wireshark pro analýzu RTP spojení. Při těchto testech se ukázalo, že zařízení generuje velice kvalitní RTP stream. Jitter při spojeních po lokální síti nepřesahoval 5 ms a největší odchylka doručení paketu 40 ms. Zařízení tak generovalo kvalitnější stream než všechny testované softwarové telefony. Pro srovnání je na obrázku 4.2 ukázka analýzy stejného úseku spojení, kde je rozdíl velice dobře patrný. Nutno však dodat, že rozdíl se prakticky neprojeví, neboť všechna zařízení implementují nějaký druh vyrovnávacího bufferu, který jitter do jisté úrovně (za cenu zvýšení zpoždění) eliminuje. Zpoždění je pak v obou případech tak malé, že větší je způsobeno právě implementovaným jitter bufferem.

[9]Velká část kódu aplikace, která nepoužívá žádné „Xilinx-specifické“ funkce ale jen POSIX rozhraní pro vlákna a sockets rozhraní pro síťovou vrstvu, je jednoduše přenositelná na linux. Část vývoje SW zařízení dokonce probíhala právě na linuxu.



Obrázek 4.2: Srovnání kvality RTP streamu zařízení se streamem VoIP softphone. Vlevo analýza odchozího RTP streamu ze zařízení, vpravo analýza streamu v opačném směru generovaného programem wxCmmunicator.

4.3 Celkové testy

Snahou při celkových testech zařízení bylo simulovat „běžný“ provoz VoIP telefonu. Jednalo se prakticky o kombinované testy SIP/SDP a RTP stacku, s kontrolním sledováním síťového provozu. Až na nevyřešené „pády“ při delších hovorech, rozebírané v části věnované testům RTP stacku, se při tomto „zkušebním provozu“ nobjevily žádné další problémy. Při těchto testech se ukázalo, že implementovaná podmnožina SIP/SDP protokolu je zcela dostačující pro obvyklé situace nastávající při peer-to-peer VoIP komunikaci. Praktické zkušenosti pak ukazují, že ani ve všech k testování využitých SIP softphone aplikacích nejsou protokoly dodržovány/kontrolovány zcela 100%. Většina z aplikací se snaží být co „nejtolerantnější“ a naopak používat co nejužší množinu protokolu.

Výjimkou je Ekiga, která schopnosti protistrany prověří opravdu důkladně, ať už používáním přípustných, ale ne tak zcela běžných postupů při komunikaci (např. změna SIP portu během dialogu), či důkladnou kontrolou všech údajů v SIP zprávách.

5 Závěr

V rámci DP byl jako SoC na přípravku ML403 úspěšně realizován VoIP telefon kompatibilní s protokoly SIP/RTP. Zařízení umožňuje dvoubodový plně duplexní přenos audio dat po počítačové síti kompatibilní s IP protokolem (lokální síť LAN, Internet, ...), do které je připojeno skrze Ethernet linkovou vrstvu. Telefon umožňuje přenos zvuku ve formátech definovaných standardy G.711 A-Law a G.711 μ-Law, tedy dvou variantách logaritmicky komprimované pulzně kódové modulace (PCM) s vzorkovací frekvencí 8 kHz.

Zpoždění zvuku způsobené zpracováním dat v zařízení nepřesahuje 20 ms, nepočítáme-li implementovaný jitter buffer pro příchozí data, který vyrovnává možné výchylky v příchozím datovém streamu vzniklé v přenosové síti. Velikost tohoto bufferu je nastavena na čtyři RTP pakety, což odpovídá 80 ms. To je hodnota, která dovoluje vyrovnat běžné výchylky při přenosu přes běžná internetová připojení typu ADSL, ale ještě nezpůsobuje subjektivně rozeznatelné zpoždění mezi přijímaným a vysílaným zvukem.

Zařízení poskytuje jednoduché uživatelské rozhraní, sestávající z LCD displeje a PS/2 klávesnice, umožňuje automatickou konfiguraci síťového připojení pomocí DHCP a obsahuje podporu pro adresaci pomocí doménových jmen.

Implementace je typu SoC (systém na chipu) a je postavena na mikroprocesoru MicroBlaze a OS Xilkernel, které jsou součástí vývojové platformy Xilinx EDK. V rámci práce byly realizovány řadiče LCD, PS/2 a upraven (doplněn o inicializační/resetovací obvod tak, aby pro svoji funkci nepotřeboval žádný další „externí“ obvod) řadič AC97. K tomuto ve VHDL navrženému/upravenému HW byla dále dopsána SW podpora (ovladače) pro OS Xilkernel. Z dalších SW modulů byly kompletně vlastními prostředky implementovány moduly pro komunikaci pomocí protokolů SIP a RTP – SIP a RTP stacky, DNS resolver pro překlad doménových jmen na IP adresy a kompletní uživatelská konzole. Kromě toho byla upravena použitá knihovna TCP/IP stacku lwip tak, aby umožňovala kompletní konfiguraci síťového připojení zařízení pomocí protokolu DHCP. Veškerá tato SW funkcionalita (včetně OS a systémových knihoven!) je „vměstnána“ do méně než 120 kB programového kódu a systém pro svůj běh potřebuje pouhé 2 MB RAM.

Zařízení bylo úspěšně otestováno proti několika stávajícím SW VoIP telefonům i SIP serveru Asterisk. Dále bylo ukázáno, že i při konfiguraci, která dovoluje provozovat navržený systém na levných FPGA typu Spartan-3E, má zařízení dostatečný výkon pro plně duplexní zpracování zvuku v reálném čase.

Slabým místem zařízení je jeho stabilita. Z neznámého důvodu dochází při delších hovorech k pádu zařízení. Ty mohou být způsobeny jak „vlastním“ kódem, tak knihovnou lwip, která se při plně duplexním (navíc vícenásobném – zároveň datovým RTP streamem musí být stále aktivní i „řídící“ SIP session) spojení a vyšší zátěži ukázala být poměrně problematickou a její nasazení v podobně „konkurenčním“ prostředí je do budoucna dobré velmi zvážit.

Podařilo-li by se výše zmíněný problém odstranit, představovalo by navržené zařízení zcela funkční a reálně použitelné řešení VoIP komunikace pro lokální firemní sítě nebo „domácí uživatele“, u kterých není potřeba „dynamická“ registrace zařízení u registračního SIP serveru.

6 Literatura

[1] AC97 Component Specification [online].
http://download.intel.com/support/motherboards/desktop/sb/ac97_r23.pdf
[cit. 2008-05-07].

[2] Comparison of VoIP software [online].
http://en.wikipedia.org/wiki/Comparison_of_VoIP_software
[cit. 2008-05-07].

[3] EDK 9.1 MicroBlaze Tutorial in Virtex-4 [online].
http://www.xilinx.com/support/techsup/tutorials/EDK_91_MB_Tutorial.pdf
[cit. 2008-05-07].

[4] EDK OS and Libraries Document Collection [online].
http://www.xilinx.com/ise/embedded/edk91i_docs/oslib_rm.pdf
[cit. 2008-05-07].

[5] LM4550B AC'97 Rev 2.1 Multi-Channel Audio Codec with Stereo Headphone Amplifier, Sample Rate Conversion and National 3D Sound [online].
http://www.national.com/ds.cgi/LM/LM4550B.pdf
[cit. 2008-05-08].

[6] lwip library v2.00.a [online].
http://japan.xilinx.com/ise/embedded/edk91i_docs/lwip_v2_00_a.pdf
[cit. 2008-05-07].

[7] MicroBlaze sell sheet [online].
http://www.xilinx.com/publications/prod_mktg/MicroBlaze_Sell_Sheet.pdf
[cit. 2008-05-07].

[8] ML403 EDK Embedded MicroBlaze Reference Design [online].
http://www.xilinx.com/products/boards/ml403/files/ml403_emb_ref_81.zip
[cit. 2008-05-07].

[9] SIP: The Never-Ending Hype Wagon [online].
http://www.dailypayload.com/2007/0618.html
[cit. 2008-05-07].

[10] Wikipedia, the free encyclopedia [online].
http://en.wikipedia.org/wiki/H323
[cit. 2008-05-08].

[11] Xilinx UG230 Spartan-3E Starter Kit Board User Guide [online].
http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf
[cit. 2008-05-07].

[12] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[13] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002.

[15] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

A Gramatika SIP parseru

Kompletní gramatika (množinu pravidel, množiny neterminálních a terminálních symbolů a počáteční symbol) jazyka SIP protokolu tak jak ji implementuje sestrojený LL1 parser (`parser.c`). Terminální a neterminální symboly jsou tvořeny slovy (kde slovo je i samostatný nealfanumerický znak), oddělovačem je mezera. Pravidla mají běžný formát zápisu: `A -> B c`.

```
        --------- TERMINALY ----------

sip : ident integer @ / version nl invite ack bye cancel udp via to
from cseq ; < > = callid ctype string contact register allbutnl

        -------- NETERMINALY ---------

SIP STATUSLINE HEADER REQUEST STATUS REQUESTMETHOD VIA TO FROM CSEQ
CALLID CTYPE UNKNOWN NAME ADDRESS PARAMETER PORT HOST DATA KEYWORD
PARAMETERVALUE VALUE ADDRESS2 INTIDENT CONTACT CIDHOST PADDRESS QADDRESS

        ----- STARTOVACI SYMBOL ------

SIP

        ---------- PRAVIDLA ----------

SIP -> STATUSLINE nl HEADER nl

STATUSLINE -> REQUEST
STATUSLINE -> STATUS

REQUEST -> REQUESTMETHOD PADDRESS sip / version
REQUESTMETHOD -> invite
REQUESTMETHOD -> ack
REQUESTMETHOD -> bye
REQUESTMETHOD -> cancel
REQUESTMETHOD -> register

STATUS -> sip / version integer DATA

HEADER -> VIA nl HEADER
HEADER -> TO nl HEADER
HEADER -> FROM nl HEADER
HEADER -> CSEQ nl HEADER
HEADER -> CALLID nl HEADER
HEADER -> CTYPE nl HEADER
HEADER -> CONTACT nl HEADER
HEADER -> UNKNOWN nl HEADER
HEADER ->

VIA -> via : sip / version / udp ident PORT PARAMETER
```

```
TO -> to : NAME ADDRESS PARAMETER
FROM -> from : NAME ADDRESS PARAMETER
CSEQ -> cseq : integer REQUESTMETHOD
CALLID -> callid : INTIDENT CIDHOST
CTYPE -> ctype : ident / ident
CONTACT -> contact : NAME ADDRESS PARAMETER
UNKNOWN -> ident : DATA

PORT -> : integer
PORT ->

PARAMETER -> ; ident PARAMETERVALUE PARAMETER
PARAMETER ->
PARAMETERVALUE -> = VALUE
PARAMETERVALUE ->

NAME -> string
NAME ->

ADDRESS -> < QADDRESS >
ADDRESS -> PADDRESS
QADDRESS -> sip : INTIDENT HOST PORT PARAMETER
PADDRESS -> sip : INTIDENT HOST PORT

HOST -> @ INTIDENT
HOST ->

CIDHOST -> @ INTIDENT
CIDHOST ->

INTIDENT -> integer
INTIDENT -> ident
INTIDENT -> KEYWORD

DATA -> allbutnl DATA
DATA ->

VALUE -> INTIDENT
VALUE -> string

KEYWORD -> sip
KEYWORD -> udp
KEYWORD -> invite
KEYWORD -> ack
KEYWORD -> bye
KEYWORD -> cancel
KEYWORD -> via
KEYWORD -> to
KEYWORD -> from
KEYWORD -> cseq
```

```
KEYWORD -> callid
KEYWORD -> ctype
KEYWORD -> contact
KEYWORD -> register
```

B Gramatika SDP parseru

Kompletní gramatika (množinu pravidel, množiny neterminálních a terminálních symbolů a počáteční symbol) jazyka SDP protokolu tak jak ji implementuje sestrojený LL1 parser (`parser.c`). Terminální a neterminální symboly jsou tvořeny slovy (kde slovo je i samostatný nealfanumerický znak), oddělovačem je mezera. Pravidla mají běžný formát zápisu: `A -> B c`.

```
        --------- TERMINALY ----------

nl m = rtp / avp ident integer c in ip4 allbutnl

        -------- NETERMINALY ---------

SDP ITEM MEDIA UNRECOGNIZED CODEC OPERATOR DATA CONNECTION

        ----- STARTOVACI SYMBOL ------

SDP

        ---------- PRAVIDLA ----------

SDP -> ITEM

ITEM -> MEDIA nl ITEM
ITEM -> CONNECTION nl ITEM
ITEM -> UNRECOGNIZED nl ITEM
ITEM ->

MEDIA -> m = ident integer rtp / avp CODEC
CONNECTION -> c = in ip4 ident
UNRECOGNIZED -> ident = DATA

CODEC -> integer CODEC
CODEC ->

DATA -> allbutnl DATA
DATA ->
```

C PS/2 keyboard driver IP core

Následující příloha je anglickou uživatelskou dokumentací PS/2 řadiče použitého v implementovaném zařízení. Text je (včetně udávaných příloh) dostupný také na WWW stránkách autora na adrese:

http://tumic.wz.cz/fel/online/36SEM/PS2/

PS/2 keyboard driver IP core

for Xilinx EDK

Martin Tůma, FEL ČVUT

Features

- Provides high-level interface for communication with a PS/2 device
- Full bidirectional PS/2 protocol support
- Provides polling mode and interrupt mode interface to the device
- Usable on all boards with PS/2 connector supported by Xilinx EDK

Introduction

This application note describes a PS/2 keyboard driver EDK IP core which handles the physical and electrical part of the PS/2 protocol and provides a command based, high-level interface for communication with a PS/2 keyboard.

Description

Synopsis

```
#include <ps2_kbd_driver.h>

void ps2_kbd_init(PS2KbdInst *instance,
  Xuint32 baseaddr, PS2KbdMode opmode)
void ps2_kbd_set_irq_handler(PS2KbdInst *instance,
  ps2_kbd_irq_handler hndlr)
Xuint8 ps2_kbd_get_code(PS2KbdInst *instance)
void ps2_kbd_set_code(PS2KbdInst *instance, Xuint8 code)
void ps2_kbd_irq_handler(void *instance)
XStatus ps2_kbd_selftest(PS2KbdInst *instance)
```

Description

The *ps2_kbd_init()* function initializes an instance of the driver for the device given by it's base address. It also sets the operation mode of the device which can by either `PS2_KBD_POLLING_MODE` or `PS2_KBD_IRQ_MODE`.

If interrupt mode set, the function *ps2_kbd_set_irq_handler* sets the user interrupt handler for the device. Received data is automaticaly stored in drivers FIFO. If you want to handle the received scancode in the interrupt handler self, you can call *ps2_kbd_get_code()* in the handler.

The *ps2_kbd_get_code()* function returns the last scancode/response command sent by the keyboard if a new scancode/command has arrived since the last function call, zero otherwise. When the received scan code is not valid (parity error), the function initializes sending a repeat command (0xFE) to the device and returns 0. If in interrupt mode, data is read from the internal FIFO.

The *ps2_kbd_set_code()* function sends a command with the value *code* to the keyboard. This function always succeeds as the host is the bus master in the PS/2 protocol. If another one send operation is still in progress while the function is called, the function waits until the previous transmission is finished.

If the device is in interrupt mode, *ps2_kbd_irq_handler()* has to be registred as the interrupt

handler for the device in Xilkernel or on the IRQ controller on standalone systems.

Design Files

The accompanying ZIP file (ps2_kbd_driver.zip) contains the following files:

Structure

- **drivers**
  - **ps2_kbd_driver_v1_00_a**
    - **data**
      - ps2_kbd_driver_v2_1_0.mdd
      - ps2_kbd_driver_v2_1_0.tcl
    - **src**
      - Makefile
      - ps2_kbd_driver.c
      - ps2_kbd_driver.h
      - ps2_kbd_driver_l.h
      - ps2_kbd_driver_selftest.c
- **pcores**
  - **ps2_kbd_driver_v1_00_a**
    - **data**
      - ps2_kbd_driver_v2_1_0.mpd
      - ps2_kbd_driver_v2_1_0.pao
    - **hdl**
      - **vhdl**
        - ps2_clk_mod.vhd
        - ps2_drv.vhd
        - ps2_kbd_driver.vhd
        - ps2_mainfsm_mod.vhd
        - ps2_rcv_mod.vhd
        - ps2_send_mod.vhd
        - user_logic.vhd

Description

ps2_kbd_driver_v2_1_0.mpd
: Microprocessor Peripheral Definition file (MPD). Describes bus and external port connections of the peripheral device.

ps2_kbd_driver_v2_1_0.pao
: Peripheral Analyse Order file (PAO). Dictates the correct order of synthesis for the VHDL source files.

ps2_kbd_driver.vhd
: Peripheral top level design. Instantiates the IPIF and user logic.

user_logic.vhd
: Connects the PS2 driver main module to the OPB throught the IPIF.

ps2_drv.vhd
: PS2 driver main module.

ps2_mainfsm_mod.vhd
: PS2 driver main FSM - switches between the transmit and the recieve module.

ps2_clk_mod.vhd
: PS2 driver clock module. Filters PS2 clock and detects falling/rising edge on it.

ps2_rcv_mod.vhd
: PS2 driver receive module. Handles incoming PS2 communication.

ps2_send_mod.vhd
: PS2 driver transmit module. Handles outgoing PS2 communication.

ps2_kbd_driver_v2_1_0.mdd
: Microprocessor Driver Definition file (MDD). Lists the peripherals that are able to use the software driver.

ps2_kbd_driver_v2_1_0.tcl
: TCL script used by EDK for `#define` statements generation (e.g. baseaddress) in the *xparameters.h* header file.

Makefile
: Makefile used for building the SW driver.

ps2_kbd_driver.c
: Software driver source file.

ps2_kbd_driver.h
: Software driver header file.

ps2_kbd_driver_l.h
: Software driver low-level macros & definitions.

ps2_kbd_driver_selftest.c
: Peripheral self-test. Contains code to test the correct operation of the peripheral.

Design Implementation

This section describes how to create a simple EDK project to see a working demo of the peripheral driver.

1. Start EDK and create a new project with the Base System Builder (BSB). You can find detailed instructions how to do this in the EDK tutorial [1]. In the peripheral selection part of the wizard leave all devices exept RS232 (will be used as STDIN and STDOUT) unchecked. Also disable the creation of all sample applications (Memory test, Peripheral SelfTest).
2. Copy both top directories (pcores, drivers) of the accompanying archive to the root directory of your new EDK project.
3. In the main EDK menu select: "Project"->"Rescan User Repositories".
4. In the "Project Information Area" select "IP Catalog". Under "Project Local pcores" you should see "PS2_KBD_DRIVER". Double click on it to add the IP to the design. A *ps2_kbd_driver_0* item should now appear in the System Assembly View window.
5. Connect the IP to the OPB bus by selecting *mp_opb* as the bus connection for the *SOPB* connector of *ps2_kbd_driver_0* in the System Assembly View window.
6. Switch the "Filters" radio button to "Ports" and connect the IP ports. Set *ps2_kbd_driver_0_PS2_clk* net for *PS2_clk* and *ps2_kbd_driver_0_PS2_data* for *PS2_data*. If you intend to use the device in interrupt mode, also connect the interrupt port to the interrupt port of the processor (or the interrupt controller when used).
7. Switch the "Filters" radio button to "Addresses" and click on the "Generate Addresses" button.
8. If other Processor-Bus clock frequency then 50 MHz is used, a matching *C_OPB_Clk_FREQ_HZ* generic value must be chosen in the IP core configuration ("System Assembly View window"->"ps2_kbd_driver_0"->"Configure IP").
9. Edit the UCF file. You can open it from the "Project Information Area" window under the "Project" bookmark. Add constraints for the PS2 clock and PS2 data pins. Name the pins *PS2_clk* and *PS2_data*. The constraints are board-specific and you can find them in the documentation for your board. For example for the Spartan-3E Starter board you need to add the following to the UCF file:

```
NET PS2_clk LOC = G14 | IOSTANDARD = LVCMOS33;
NET PS2_data LOC = G13 | IOSTANDARD = LVCMOS33;
```

10. Edit the MHS file. Add the following two lines at the end of the file:

```
PORT PS2_clk = ps2_kbd_driver_0_PS2_clk, DIR=IO
PORT PS2_data = ps2_kbd_driver_0_PS2_data, DIR=IO
```

11. In the main menu click on "Hardware"->"Generate Bitstream". The HW part of the project should now successfully synthetize.
12. In the "Project Information Area" select "Applications" and click on "Add Software Application Project" to create a new SW project.
13. Add a new source to the created SW project with the following content:

```
#include "xparameters.h"
#include "ps2_kbd_driver.h"

void main() {
    PS2KbdInst kbd;

    ps2_kbd_init (
        &kbd,
        XPAR_PS2_KBD_DRIVER_0_BASEADDR,
        PS2_KBD_POLLING_MODE
    );

    ps2_kbd_selftest(&kbd);
}
```

14. Select the SW project to initialize the BRAM in the "Project Information Area". Disable any other application (microblaze_0_bootloop, microblaze_0_xmdstub) to initialize the BRAM.
15. In the EDK main menu select "Software"->"Build All User Applications". The SW project should successfully compile.
16. Select "Device Configuration"->"Download Bitstream" to program the device.

A keyboard connected to the FPGA board should blink with its leds like start lights as soon as the FPGA finishes its configuration. Informations about the process of the selftest are also writen to STDOUT (RS232).

Implementation notes

Although the IP core/driver was primary designed for a ps2 keyboard, it can be used (except the self-test function of course) for any PS2 device, for example a mouse, since it only handles byte transfers from/to device which are independent on the upper layer protocol.

References

[1] http://www.xilinx.com/support/techsup/tutorials/edk_tutorials.htm

Atachments

1. ps2_kbd_driver.zip - A ZIP archive containing the sources

D LCD driver EDK IP core

Následující příloha je anglickou uživatelskou dokumentací LCD řadiče použitého v implementovaném zařízení. Text je (včetně udávaných příloh) dostupný také na WWW stránkách autora na adrese:

http://tumic.wz.cz/fel/online/36SEM/LCD/

LCD driver EDK IP core

for Xilinx EDK

Martin Tůma, FEL ČVUT

Features

- Provides interface for communication with a LCD in EDK
- Uses the LCDs 4-bit data interface used on most Xilinx development boards
- Most functionality implemented in VHDL (low CPU usage)

Introduction

This application note describes a Sitronix ST7066U LCD controller driver for EDK which provides a command based, high-level interface for communication with a LCD.

Description

Synopsis

```
#include <lcd_driver.h>

void lcd_init(LCDInst *instance, Xuint32 baseaddr)
void lcd_write_cmd(LCDInst *instance, Xuint8 cmd)
void lcd_write_data(LCDInst *instance, Xuint8 data)
```

Description

The *lcd_init()* function initializes an instance of the driver for the device given by it's base address. It must be called befor any operation on the device.

The *lcd_write_cmd()* function writes an instruction to the LCD controller. For a list of available instructions see the LCD controller datasheet [1] or the documentation of your development board. The instruction code consist only of the data bits value, the RW and RS bits are set automaticaly by the driver/IP core. The header file of the driver contains `#define`s for the most of the available instructions.

The *lcd_write_data()* function writes data to the controllers CG RAM or DD RAM (depends on a previous instruction).

Note: There is no support for the "Read Busy Flag and Address" and "Read Data from CG RAM or DD RAM" instructions, as the driver/IP core is conceived for write-only operation mode.

Design Files

The accompanying ZIP file (lcd_driver.zip) contains the following files:

Structure

- **drivers**
  - **lcd_driver_v1_00_a**
    - **data**
      - lcd_driver_v2_1_0.mdd
      - lcd_driver_v2_1_0.tcl

- **src**
  - Makefile
  - lcd_driver.c
  - lcd_driver.h
  - lcd_driver_l.h
  - lcd_driver_selftest.c
- **pcores**
  - **ps2_kbd_driver_v1_00_a**
    - **data**
      - lcd_driver_v2_1_0.mpd
      - lcd_driver_v2_1_0.pao
    - **hdl**
      - **vhdl**
        - lcd.vhd
        - lcd_cmd.vhd
        - lcd_driver.vhd
        - lcd_fsm.vhd
        - lcd_init.vhd
        - user_logic.vhd

Description

lcd_driver_v2_1_0.mpd
: Microprocessor Peripheral Definition file (MPD). Describes bus and external port connections of the peripheral device.

lcd_driver_v2_1_0.pao
: Peripheral Analyse Order file (PAO). Dictates the correct order of synthesis for the VHDL source files.

lcd_driver.vhd
: Peripheral top level design. Instantiates the IPIF and user logic.

user_logic.vhd
: Connects the LCD driver main module to the OPB throught the IPIF.

lcd.vhd
: LCD driver main module.

lcd_fsm.vhd
: LCD driver main FSM.

lcd_cmd.vhd
: Command transfer module. Handles the transfer of an instruction/data byte using two sequential 4-bit operations.

lcd_init.vhd
: Power-On initialization module. Establishes the required communication protocol.

lcd_driver_v2_1_0.mdd
: Microprocessor Driver Definition file (MDD). Lists the peripherals that are able to use the software driver.

lcd_driver_v2_1_0.tcl
: TCL script used by EDK for `#define` statements generation (e.g. baseaddress) in the *xparameters.h* header file.

Makefile
: Makefile used for building the SW driver.

lcd_driver.c
: Software driver source file.

lcd_driver.h
: Software driver header file.

lcd_driver_l.h
: Software driver low-level definitions & macros.

lcd_driver_selftest.c
: Peripheral self-test. Contains code to test the correct operation of the peripheral.

Design Implementation

This section describes how to create a simple EDK project to see a working demo of the peripheral driver.

1. Start EDK and create a new project with the Base System Builder (BSB). You can find detailed instructions how to do this in the EDK tutorial [2]. In the peripheral selection part of the wizard leave all devices exept RS232 (will be used as STDIN and STDOUT) unchecked. Also disable the creation of all sample applications (Memory test, Peripheral SelfTest).
2. Copy both top directories (pcores, drivers) of the accompanying archive to the root directory of your new EDK project.
3. In the main EDK menu select: "Project"->"Rescan User Repositories".
4. In the "Project Information Area" select "IP Catalog". Under "Project Local pcores" you should see "LCD_DRIVER". Double click on it to add the IP to the design. A *lcd_driver_0* item should now appear in the System Assembly View window.
5. Connect the IP to the OPB bus by selecting *mp_opb* as the bus connection for the *SOPB* connector of *lcd_driver_0* in the System Assembly View window.
6. Switch the "Filters" radio button to "Ports" and connect the IP ports. Set *lcd_driver_0_LCD_E* net for *LCD_E*, *lcd_driver_0_LCD_RS* for *LCD_RS*, *lcd_driver_0_LCD_RW* for *LCD_RW* and *lcd_driver_0_LCD_DATA* for *LCD_DATA*.
7. Switch the "Filters" radio button to "Addresses" and click on the "Generate Addresses" button.
8. If other Processor-Bus clock frequency then 50 MHz is used, a matching *C_OPB_Clk_FREQ_HZ* generic value must be chosen in the IP core configuration (System Assembly View window->lcd_driver_0->Configure IP).
9. Edit the UCF file. You can open it from the "Project Information Area" window under the "Project" bookmark. Add constraints for all LCD_DRIVER pins. Name the pins *LCD_E*, *LCD_RW*, *LCD_RS* and *LCD_DATA*. The constraints are board-specific and you can find them in the documentation for your board. For example for the Spartan-3E Starter board you need to add the following to the UCF file:

```
NET LCD_E LOC = M18 | IOSTANDARD = LVCMOS33;
NET LCD_RS LOC = L18 | IOSTANDARD = LVCMOS33;
NET LCD_RW LOC = L17 | IOSTANDARD = LVCMOS33;
NET LCD_DATA<0> LOC=R15 | IOSTANDARD = LVCMOS33;
NET LCD_DATA<1> LOC=R16 | IOSTANDARD = LVCMOS33;
NET LCD_DATA<2> LOC=P17 | IOSTANDARD = LVCMOS33;
NET LCD_DATA<3> LOC=M15 | IOSTANDARD = LVCMOS33;
```

10. Edit the MHS file. Add the following four lines at the end of the file:

```
PORT LCD_E = lcd_driver_0_LCD_E, DIR=O
PORT LCD_RW = lcd_driver_0_LCD_RW, DIR=O
PORT LCD_RS = lcd_driver_0_LCD_RS, DIR=O
PORT LCD_DATA = lcd_driver_0_LCD_DATA, DIR=O, VEC=[3:0]
```

11. In the main menu click on "Hardware"->"Generate Bitstream". The HW part of the project should now successfully synthetize.
12. In the "Project Information Area" select "Applications" and click on "Add Software Application Project" to create a new SW project.
13. Add a new source to the created SW project with the following content:

```
#include "xparameters.h"
#include "lcd_driver.h"

void main() {
    LCDInst lcd;

    lcd_init(&lcd, XPAR_LCD_DRIVER_0_BASEADDR);
    lcd_selftest(&lcd);
}
```

14. Select the SW project to initialize the BRAM in the "Project Information Area". Disable any other application (microblaze_0_bootloop, microblaze_0_xmdstub) to initialize the BRAM.
15. In the EDK main menu select "Software"->"Build All User Applications". The SW project should successfully compile.
16. Select "Device Configuration"->"Download Bitstream" to program the device.

As soon as the FPGA finishes its configuration, "LCD test" should appear on the LCD. Informations about the process of the selftest are also writen to STDOUT (RS232).

References

[1] http://www.sitronix.com.tw/sitronix/product.nsf/Doc/ST7066U?OpenDocument
[2] http://www.xilinx.com/support/techsup/tutorials/edk_tutorials.htm

Atachments

1. lcd_driver.zip - A ZIP archive containing the sources

E Xilinx lwip DHCP mini HOWTO

Následující příloha je anglickou uživatelskou dokumentací k patchi knihovny lwip a jeho použití v Xilinx EDK/SDK. Text je (včetně udávaných příloh) dostupný také na WWW stránkách autora na adrese:

http://tumic.wz.cz/it/online/lwip_dhcp/

Xilinx lwip DHCP mini HOWTO

Martin Tůma, FEE CTU

Introduction

This paper describes how to enable address allocation throught DHCP for applications developed with Xilinx Platform Studio using Xilinx's lwip library.

Description

How to make it work

The lwip v2.00.a library provided by Xilinx (at least in EDK 9.1) does not enable to use the in the lwip library included DHCP support by default. A little 'hack' is needed to make DHCP work - the lwip Makefile (`Makefile_mb` for Microblaze and `Makefile_ppc` for PowerPC) in `$EDK/sw/ThirdParty/sw_services/lwip_v2_00_a/src` has to be changed. Add `$(LWIPDIR)/core/dhcp.c` to the file list defined under `COREFILES`. For the MicroBlaze lwip Makefile version you can use the Makefile in atachment 1.

There is no support for obtaining the DNS server within the DHCP session by default. If you need to get the DNS server address, you can use the in the atachment also included patched files `dhcp.c`, `dhcp.h` and `netif.h`. Your `netif` structure will then include an additional entry of the type `struct ip_addr` called `dns` which will contain the DNS server IP address. The file `dhcp.c` belongs to `($LWIPDIR)/lwip/src/core`, the files `netif.h` and `dhcp.h` to `($LWIPDIR)/lwip/src/include/lwip`.

The second step is to make your application correctly use the lwip DHCP functions. The example below shows the essential code needed in an application that uses DHCP to configure its network interface:

```
#include <netif/xemacliteif.h>
#include <xmk.h>
#include <xparameters.h>

#include <lwip/mem.h>
#include <lwip/tcpip.h>
#include <lwip/dhcp.h>


#define DEBUG

#ifdef DEBUG
#define DEBUG_MSG(msg) xil_printf(msg);
#else
#define DEBUG_MSG(msg)
#endif

#define MAC_ADDR            {0x01, 0x02, 0x03, 0x04, 0x05, 0x06}
#define DHCP_TIMEOUT_MS     2000

extern XEmacLiteIf_Config XEmacLiteIf_ConfigTable[];
static struct netif *app_netif;


void* netif_config_thread(void *arg)
{
    unsigned char macaddr[6] = MAC_ADDR;
    struct ip_addr ipaddr, netmask, gateway;
    int mscnt = 0;

    XEmacLiteIf_Config *xemacif_ptr = &XEmacLiteIf_ConfigTable[0];

    // Set up the MAC address for the ethernet MAC
```

```
    xemacliteif_setmac(0, (u8_t *)macaddr);

    // Clear the IP address, Subnet Mask, and Gateway
    ipaddr.addr = 0;
    netmask.addr = 0;
    gateway.addr = 0;

    // Set up the lwIP network interface
    // Allocate and configure the app's netif
    app_netif = (struct netif *) mem_malloc(sizeof(struct netif));
    if(app_netif == NULL)
    {
        DEBUG_MSG("ERROR: Out of memory for default netif\r\n");
        return;
    }
    app_netif = netif_add(app_netif, &ipaddr, &netmask, &gateway,
      &XEmacLiteIf_ConfigTable[0], xemacliteif_init, tcpip_input);
    netif_set_default(app_netif);

    // Register the XEMAC interrupt handler with the controller
    // and enable interrupts within XMK
    register_int_handler(XPAR_OPB_INTC_0_ETHERNET_MAC_IP2INTC_IRPT_INTR,
      (XInterruptHandler)XEmacLite_InterruptHandler, xemacif_ptr->instance_ptr);
    enable_interrupt(XPAR_OPB_INTC_0_ETHERNET_MAC_IP2INTC_IRPT_INTR);

    // Start the DHCP "Client Daemon"
    DEBUG_MSG("Starting DHCPCD...\r\n");
    dhcp_start(app_netif);

    while (1) {
        sleep(DHCP_FINE_TIMER_MSECS);
        dhcp_fine_tmr();
        mscnt += DHCP_FINE_TIMER_MSECS;
        if (mscnt >= DHCP_COARSE_TIMER_SECS*1000) {
            dhcp_coarse_tmr();
            mscnt = 0;
        }
    }

}

void* socket_thread(void* arg)
{
    int mscnt = 0;

    // Initialize the lwIP library
    DEBUG_MSG("Initializing the lwIP library...\r\n");
    lwip_init();
    sleep(200);
    DEBUG_MSG("lwIP initialization done\r\n");

    // Start network configuration (+DHCPCD) thread
    DEBUG_MSG("Configuring IP...\r\n");
    sys_thread_new ((void (*)(void*))netif_config_thread, 0, 0);

    // Wait for DHCP IP configuration
    while (1) {
        sleep(DHCP_FINE_TIMER_MSECS);
        if (app_netif->ip_addr.addr) {
            DEBUG_MSG("DHCP request success\r\n");
            break;
        }
        mscnt += DHCP_FINE_TIMER_MSECS;
        if (mscnt >= DHCP_TIMEOUT_MS) {
            DEBUG_MSG("ERROR: DHCP request timed out\r\n");
            return;
        }
    }
    DEBUG_MSG("IP configuration done\r\n");


    // START THE APPLICATION THREAD(S)
    // sys_thread_new ((void (*)(void*)) app_thread, 0, 1);
    // ...
}


int main()
{
```

```
    // Launch XMK
    DEBUG_MSG("System initialization start\r\n");
    xilkernel_main();

    return 0;
}
```

Fig. 1.: Simple application using DHCP network interface configuration

Note: The `socket_thread` thread must be defined in the Xilkernel configuration to be started at Xilkernel start.

How it works

After initializing the interface, the function `dhcp_start(struct netif *netif)` is called to configure the interface. Then every `DHCP_FINE_TIMER_MSECS` miliseconds the function `dhcp_fine_tmr()` and every `DHCP_COARSE_TIMER_SECS` seconds the function `dhcp_coarse_tmr()` has to be called.

More info can be found in the lwip `dhcp.c` source file and the lwip wiki [1].

References

[1] http://lwip.scribblewiki.com/DHCP

Atachments

1. lwip_patch.zip - DHCP patch archive
2. example.c - Simple DHCP using application example

F Struktura obsahu přiloženého CD

```
.
|-- DOC
|   |-- PDF
|   `-- src
|-- EDK_project
|   |-- EDK_fpgasip_v4
|   `-- EDK_fpgasip_loopback
|-- Readme.txt
|-- SW
|   `-- SIP2
|-- ace
|   `-- system.ace
|-- lwip_patch
|   |-- Makefile_mb
|   |-- dhcp.c
|   `-- netif.h
`-- peripherals
    |-- AC97
    |-- LCD
    `-- PS2
```

DOC – podadresář `PDF` obsahuje kompletní text DP ve formátu PDF, podadresář `src` pak LaTeXové „zdrojové kódy“ a obrázky použité v textu DP. PDF je možné ze zdrojů sestavit pomocí obsaženého Makefile programem make.

EDK_project – podadresář `EDK_fpgasip_v4` obsahuje kompletní projekt pro EDK 9.1 a desku ML403 včetně SDK SW projektu SIP2 s vlastní SW implementací v podadresáři `SDK_projects`. Součástí projektu jsou dvě testovací SW aplikace – `TestApp_memory`, která slouží k otestování funkčnosti přístupu do RAM[1] a `TestApp_Peripherals`, která slouží k otestování funkčnosti vlastních řadičů periferií (PS/2, LCD, AC97).
Podadresář `EDK_fpgasip_loopback` obsahuje verzi projektu pro Spartan-3E starter kit board s upraveným SW, který zvuk nepřehrává (deska nemá audio kodek), ale přijatá audio data posílá zpět protistraně.

Readme.txt – popis struktury CD, obsahuje tento text.

SW – obsahuje samotný SW projekt – vlastní VoIP aplikaci.

ace – obsahuje soubor `system.ace`, který je možné nahrát na flash kartu, ze které lze poté inicializovat desku ML403. Obsahem souboru je kompletní HW a SW zařízení, po zapnutí desky se „samo“ nakonfiguruje FPGA a program se nahraje do RAM a spustí. Po zapnutí desky s vloženou flash kartou s tímto souborem je tedy zařízení plně funkční a připravené k provozu.

lwip_patch – adresář s modifikovanými soubory knihovny `lwip` přidávajícími do knihovny podporu pro DHCP.

peripherals – obashuje samostatně vlastní/modifikované řadiče periferií (včetně ovladačů), které vznikly v rámci DP.

[1] Při syntéze HW se v EDK 9.1 může stát, že syntéza proběhne úspěšně ale přístup do RAM nefunguje.