VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

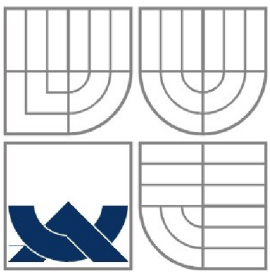
TCP/IP PROTOKOL PRO VÝUKOVOU HW/SW PLATFORMU FITKIT

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

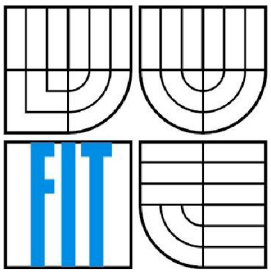
AUTOR PRÁCE
AUTHOR

JIŘÍ LESKOVEC

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TCP/IP PROTOKOL PRO VÝUKOVOU HW/SW PLATFORMU FITKIT

TCP/IP FOR HW/SW TEACHING PLATFORM FITKIT

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ LESKOVEC

VEDOUCÍ PRÁCE
SUPERVISOR

Dr. Ing. OTTO FUČÍK

BRNO 2009

Abstrakt

Bakalářská práce pojednává o zpracování TCP/IP protokolu pro výukové zařízení FITKit. Zabývá se výběrem vhodného řešení, modifikaci nalezeného řešení a provázání s cílovým zařízením. Dále popisuje finální testování a chování celého systému.

Klíčová slova

TCP/IP, vestavěné zařízení, FPGA, spinet, enc28j60, síťový protokol, fitkit

Abstract

Bachelor thesis treats of handling TCP/IP protocol for teaching device FITKit. Deals with choose of appropriate solution, modification of found solution and linking with target device. Next time describing final testing and behavior whole system.

Keywords

TCP/IP, embedded device, FPGA, spinet, enc28j60, network protocol, fitkit

Citace

Jiří Leskovec: TCP/IP PROTOKOL PRO VÝUKOVOU HW/SW PLATFORMU FITKIT, bakalářská práce, Brno, FIT VUT v Brně, 2009

TCP/IP protokol pro výukovou HW/SW platformu FITKit

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Dr. Ing. Otto Fučíka
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Leskovec
1.1.2009

Poděkování

Děkuji Dr. Ing. Otto Fučíkovi za odbornou pomoc při tvorbě práce, konzultace a rady týkající se vytváření technické dokumentace.

Obsah

1 Úvod

Tématem práce bylo rozšíření HW/SW platformy FITKit o TCP/IP protokol. Protože ethernetové rozhraní již bylo vytvořeno v předchozí bakalářské práci, bylo řešení použito jako základ této práce. Zajišťuje totiž vlastní nízkoúrovňovou komunikaci a přenos dat.

Rodina protokolů TCP/IP se skládá z protokolů ARP, UDP, IP, TCP a ICMP. Všechny protokoly jsou softwarové a kromě UDP byly protokoly implementovány v MCU a zkušebně otestovány za pomocí rozhraní SPInet (viz dále). V druhé kapitole nalezneme teoretický rozbor řešení problému a stručný popis síťové komunikace, včetně popisu jednotlivých protokolů.

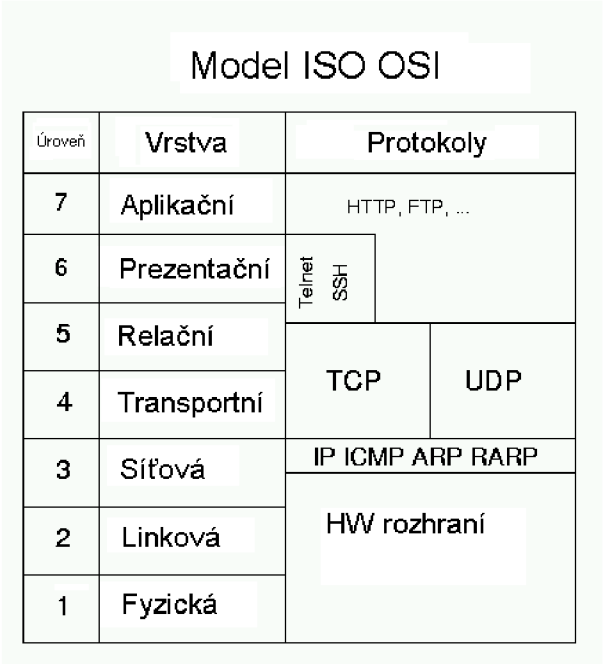
Třetí kapitola obsahuje popis možného řešení a použité nástroje a metody k zajištění co nejširší možné funkčnosti. Popisuje využití nalezeného řešení, dále popisuje funkce umožňující práci s SDRAM pamětí a implementaci jednoduché správy paměti v SDRAM. Dále pak obsahuje popis knihovny FIFO a stručný popis jejich funkcí.

Ve čtvrté kapitole nalezneme informace týkající se vlastní realizace protokolů, postup řešení, popis vyskytnutých problémů, které omezují plné využívání programu a také základní informace k ladění.

V závěru je zmíněn jednoduchý testovací program a shrnutí poznatků práce, spolu s možným rozšířením programu pro FITKit a možným pokračováním práce.

2 Popis protokolů

Sítě a síťové protokoly hrají v dnešní době velmi důležitou roli. Využívají se ve velké většině počítačového odvětví a jejich použití je všestranné. Od základní terminálové komunikace, přes mobilní zařízení až po komplexní DB a informační systémy. Síťová komunikace je založena na modelu ISO–OSI, který může být ještě dále zjednodušen na model TCP/IP. Jednotlivé vrstvy modelu se procházejí od nejvyšší po nejnižší, kde každá vrstva přidá svá data a předá je nižší. Po odeslání dat druhé straně, se zase postupuje od nejnižší po nejvyšší vrstvu a každá vrstva odebírá svá data.



Obr 1: Model ISO OSI a některé protokoly

Rodina protokolů TCP/IP sestává z protokolů ARP, ICMP, IP, TCP a UDP, které pracují na síťové, a transportní vrstvě (3. a 4. vrstva modelu). Jejich implementace je softwarová a musí být nezávislá na fyzické vrstvě, která má za úkol pouze data přenést. Protokoly ARP, ICMP a IP jsou naprogramovány ve verzi 4. Modernější IPv6 a ICMPv6 mohou být vytvořeny v některé další verzi sady TCP/IP.

2.1 ARP

Protokol ARP (Address Resolution Protocol) slouží k vyhledání linkové MAC adresy podle zadané IP adresy. Ta se poté uloží do ARP cache a využívá se při další komunikaci. Implementace je

omezena pouze na zpracování ARP paketu a generování odpovědi. Pokud přijde dotaz z adresy, která ještě není uložena v cache, systém vyšle na zadanou adresu požadavek a pakety do doby odpovědi zahazovány.

	0 – 7	8 – 15	16 – 31
0	Číslo HW protokolu (HTYPE)		Číslo protokolu (PTYPE)
32	Délka HW adresy (HLEN)	Délka log. adresy (PLEN)	Druh operace (OPER)
64	HW adresa odesílatele (prvních 32 bitů)		
96	HW adresa odesílatele (posledních 16 bitů)		Logická adresa odesílatele (prvních16 bitů)
128	Logická adresa odesílatele (posledních 16 bitů)		HW adresa příjemce (prvních 16 bitů)
160	HW adresa příjemce (posledních 32 bitů)		
192	Logická adresa příjemce		

Obr 2: Formát hlavičky ARP paketu[1]

2.2 ICMP

ICMP (Internet Control Message Protocol) je nedílnou součástí síťových protokolů. Slouží k výměně různých provozních a chybových zpráv např. nedostupnosti sítě, nedostupnosti klienta, expiraci TTL a další. Zde je implementována pouze odpověď na příkaz ping (echo reply), která zajišťuje základní ověření komunikace.

2.3 UDP

UDP (User Datagram Protocol) protokol je bezstavový. Je určen pro použití tam, kde není nutné ověřovat pořadí a správnost dat, a kde není nutné ustanovit spojení. Vhodné použití je tedy např. streamované video či zvuk, některé protokoly vyšších vrstev (TFTP, DNS) a třeba také u počítačových her. Hlavička paketu se oproti TCP skládá pouze zdrojového a cílového portu, délky dat a kontrolního součtu. Ten je také možné v rámci časové úspory možno vynechat.

2.4 IP

IP (Internet protocol) tvoří základ dnešních sítí, zejména pak internetu. Poskytuje nespojované a nespolehlivé doručování. Ve většině případů to však není na závadu, protože nad IP pracuje jiná vrstva (zpravidla TCP), která zajišťuje spolehlivost a opravy. Každé zařízení musí mít v rámci sítě unikátní adresu, která sestává ze 4 oktetů, na jejímž základě probíhá veškerá komunikace. Zařizuje také fragmentaci paketů a jejich opětovné sestavení. V současné době je nejpoužívanější verze 4

(IPv4). Z důvodu velkého rozšiřování síťových zařízení však začínají adresy docházet, a proto vznikla v nedávné době nová verze protokolu (IPv6), která poskytuje mnohem větší adresní rozsah. Většina OS je připravena na přechod na novější verzi IPv6. Je však nutné, aby byl protokol podporovaný i ve všech síťových zařízeních. Ve FITKitu byl implementován protokol IPv4. Vzhledem k nedostatku paměti v MCU nebylo možné implementovat ve FITKitu fragmentaci. Byl by potřeba další buffer stejné velikosti jako příchozí paket. Na obrázku je znázorněna hlavička paketu, ze které se získávají veškeré potřebné informace.

<table>
<tr><th></th><th>0 – 3</th><th>4 – 7</th><th>8 – 15</th><th>16 – 18</th><th>19 – 31</th></tr>
<tr><td>0</td><td>Verze protokolu</td><td>Délka hlavičky</td><td>Typ služby</td><td colspan="2">Celková délka dat</td></tr>
<tr><td>32</td><td colspan="3">Identifikace</td><td>Příznaky</td><td>Offset</td></tr>
<tr><td>64</td><td colspan="2">TTL (Time to live)</td><td>Protokol</td><td colspan="2">Kontrolní součet</td></tr>
<tr><td>96</td><td colspan="5">Zdrojová adresa</td></tr>
<tr><td>128</td><td colspan="5">Cílová adresa</td></tr>
<tr><td>160</td><td colspan="5">Nastavení (nepovinné)</td></tr>
<tr><td>160 nebo 192</td><td colspan="5">Data</td></tr>
</table>

Obr 3: Formát hlavičky IPv4 paketu[2]

2.5 TCP

TCP (Transmission control protocol) pracuje nad IP a zajišťuje spolehlivý a spojovaný přenos. Potvrzuje také správné pořadí došlých dat a také validitu dat. Ta je zajišťována kontrolním součtem, který se počítá z pseudohlavičky a je součástí hlavičky paketu. Pseudohlavička se skládá z čísla protokolu, délky dat a samotných dat. Validita dat je zajištována SYN – ACK mechanismem. Jedna strana zašle balíček dat s náhodně generovaným sekvenčním číslem (sequence number), ze kterého se po dobu spojení vychází a druhá strana zašle potvrzení příchozích dat, kde k sekvenčnímu číslu je přípočtena délka dat.

Navázání spojení je realizováno tzv. 3–way handshakingem. Aktivní strana zašle synchronizační paket (SYN), pasivní strana zašle potvrzení synchronizace a vlastní synchronizaci, na závěr je opět potvrzena aktivní stranou ACK paketem.

Uzavření spojení se provádí tzv. 4–way handshakingem. Strana, která chce uzavřít spojení, zašle paket s příznakem FIN. Druhá strana odpoví ACK paketem a hned za ním následuje paket

s příznakem FIN, který je opět druhou stranou potvrzen ACK paketem. Pro ilustraci je přiložen obrázek se strukturou hlavičky TCP paketu. Schéma TCP/IP automatu je znázorněno na obr. 3.1.1

	0 – 3	4 – 7	8 – 15	16 – 31
0	Zdrojový port			Cílový port
32	Sekvenční číslo			
64	Potvrzovací číslo			
96	Offset	Rezervováno	CWR | ECE | URG | ACK | PSH | RST | SYN | FIN	Velikost okna
128	Kontrolní součet			Ukazatel na urgentní data
160	Nastaveni (nepovinné)			
160 nebo 192	Data			

Obr 4: Formát hlavičky TCP paketu[3]

3 Návrh řešení

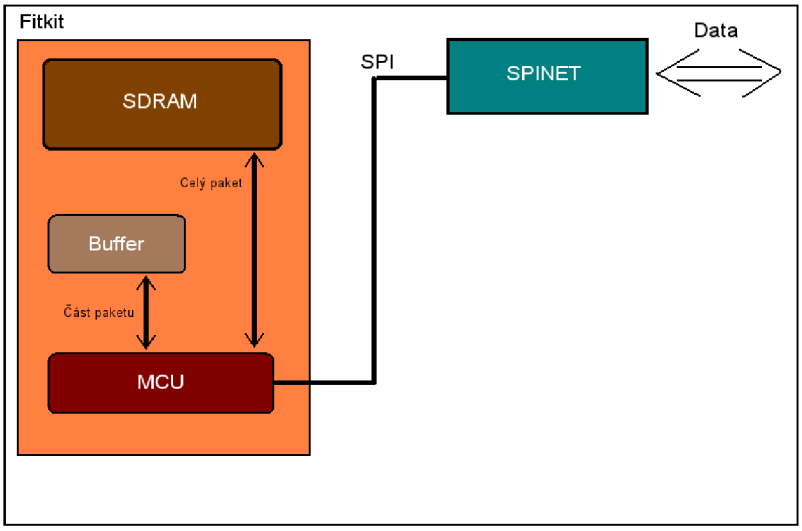
Celá práce byla stavěna tak, aby bylo v budoucnu možné rozšíření FITKitu o vestavěný ethernetový kontrolér. Toto rozšíření spočívá v přímém použití paměti SDRAM, kam by se rovnou ukládala příchozí i odchozí data a odkud by je následně kontrolér přímo přenášel. Pokud by tato situace nastala, je nejdříve nutné nastavit několik definic. Nejprve nastavit velikost bufferů pro data a poté konfigurační hodnoty. Ty sestávají převážně z adres a jsou to:

- SDRAM_PHYS_FLAGS_ADDR – adresa v SDRAM, kde jsou nastavovány fyzické příznaky
- SDRAM_INCOMING_PACKET_BUFFER_BASE – adresa, kde začíná buffer pro příchozí pakety
- SDRAM_OUTGOING_PACKET_BUFFER_BASE – adresa, kde začíná buffer pro příchozí pakety

Protože veškeré operace jsou softwarového typu a je využívána paměť v MCU, která je limitována velikostí 2 kB, bylo nutné minimalizovat veškeré paměťové operace. Nejvhodnější bylo tedy použít již hotové, otestované a paměťově nenáročné řešení.

Jako ideální se jevilo řešení **uip**[4], které bylo nutné poupravit pro potřeby FITKitu. Dále bylo nutné nalézt řešení, jak celý komplex otestovat. Vzhledem k tomu, že FITKit neobsahuje vestavěný ethernetový kontrolér, muselo se použít externí zařízení **SPINET** od firmy ASIX[5]. O tom se pojednává v jiné BP studenta Bc. René Kolaříka[6], proto se zde dále zmíním jen stručně.

Požadovanou činnost programu znázorňuje následující blokové schéma:



Obr 5: Schéma činnosti TCP/IP

Data přicházejí ze sítě do spinetu, který je ukládá do své interní paměti. Nastavuje přitom své interní příznaky a registry, zejména ukazatele na nová data. Tyto registry jsou následně cyklicky čteny v MCU přes SPI rozhraní FITKitu. Pokud existují nová data, jsou přenesena po částech do lokálního bufferu. Odtud jsou poté přesunuta do FIFO fronty v paměti SDRAM, kde se celý paket, který může být libovolné velikosti, skládá zase dohromady (voláním funkce, která přidává data po částech). Pokud nejsou žádná další data v spinetu, zpracovávají se data ve FIFO frontě. Pokud je velikost paketu větší, než je velikost vyrovnávací paměti, musí se zpracování provádět po částech.

3.1 uIP

uIP je volně šiřitelný softwarový TCP/IP stack (fronta určená k zachycení a následné zpracování dat). Je určený zejména pro vestavěné systémy, nezávislý na platformě a schopný pracovat i na 8 bitových zařízeních. Má široké možnosti volitelné konfigurace a poskytuje celou škálu funkcí. Hlavními přednostmi uIP jsou:

- Malý generovaný kód
- Nízká potřeba RAM paměti
- Libovolný počet spojení (konfigurovatelné při překladu)
- Libovolný počet portů
- Implementace protokolů podle RFC dokumentů

Kód tvoří několik souborů, které však nejsou vzájemně příliš provázané a dají se použít i jako stavební bloky. uIP obsahuje mimo TCP/IP „stacku“ také funkce pro jednoduché přesměrování paketů v uIP zařízeních, ARP tabulku a také knihovnu protosockets[1]. Tyto bloky mohou, avšak nemusí být využity. Zde byl separován pouze TCP/IP protokol a byl poupraven podle potřeb FITKitu. Dále byl nově vytvořen protokol ARP, který je zjednodušený a paměťově úspornější oproti ARP již implementovanému v uIP. Hlavní součástí je však soubor uip.c, který obsahuje kompletní kód pro zpracování paketů a správu TCP a UDP připojení.

Pro správnou činnost uIP je však nutné naprogramovat nízkoúrovňové funkce zajišťující přenos dat. Pokud používáme spinet, můžeme s úspěchem použít funkce z knihovny enc28j60 (viz kapitola 3.2.1)

V programu je definováno několik příznaků, které jsou nastavovány dle vyskytnutých událostí. Tyto příznaky jsou uloženy v globální proměnné a tedy použitelné v celém programu. Jedná se o:

1 Protosockets je knihovna podobná BSD soketům.použitá v uIP

ACKDATA	Indikuje potvrzení odeslaných dat a informuje aplikaci, že může odeslat data
NEWDATA	Informuje o nově příchozích datech
REXMIT	Upozorňuje aplikaci, že má odeslat předešlá data
POLL	Událost, která ověří, zda aplikace nemá TCP data k odeslání
CLOSE	Host uzavřel spojení, nebo aplikace se pokouší čístě uzavřít spojení
ABORT	Host přerušil spojení, nebo aplikace se pokouší přerušit spojení
CONNECTED	Indikace úspěšně sestaveného spojení
TIMEDOUT	Připojení bylo zrušeno kvůli mnoha neúspěšným pokusům
UDP_POLL	Událost, která ověří, zda aplikace nemá UDP data k odeslání

Tab 1: Výpis dostupných příznaků a jejich popis

Při každé, takto vyskytnuté události, se volá uživatelem definovaná funkce, která se nastavuje v hlavičkovém souboru uip.h. Ta by měla obsahovat obsluhu každého takového stavu. Pokud však stav obsluhovat nechceme, stačí ho prostě ignorovat a obsluhu nepsat. Není to však ideální řešení. Uživatel by měl být minimálně informován, že se něco přihodilo.

Poslední věcí, která se musí nadefinovat jsou události. Jedná se v podstatě o události, které se volají asynchronně s během programu. Jsou to tyto:

- EVENT_DATA – byla načtena nová data z HW
- EVENT_TIMER – nastala událost časovače
- EVENT_POLL – oznámení aplikaci, že máme odchozí data, která mají být odeslána

Většina ostatních možností uIP je nastavitelná v souboru uip_conf.h. Zde je nutné nastavit 8–i a 16–i bitový neznaménkový datový typ (v případě jazyka C je to „unsigned char“ a „unsigned short“ a díle všechny konfigurovatelné hodnoty:

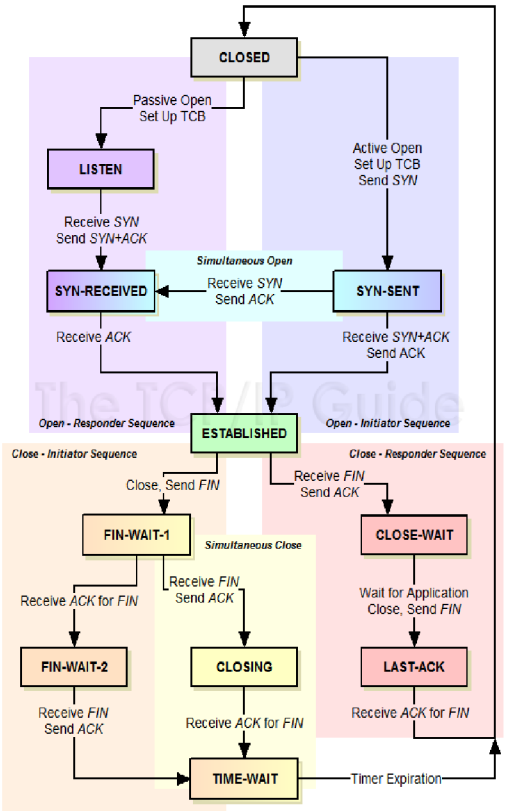
- UIP_CONF_MAX_CONNECTIONS – maximální počet připojení
- UIP_CONF_MAX_LISTENPORTS – maximální počet portů k naslouchání
- UIP_CONF_BUFFER_SIZE – velikost bufferu pro pakety
- UIP_CONF_BYTE_ORDER – velmi důležité – v jakém pořadí jsou na platformě ukládána data (little endian, big endian)
- UIP_CONF_LOGGING – podpora pro logování událostí

- UIP_CONF_UDP – kompilace UDP do programu
- UIP_CONF_UDP_CHECKSUMS – podpora kontrolního součtu pro UDP pakety
- UIP_CONF_STATISTICS – ukládá statistiku programu (počet nepřijatých paketu, počet zpracovaných TCP, ICMP a dalších paketů, ...)

Dále bylo nutné implementovat hodiny a časovače. Ty se tu nacházejí v omezené formě. Využívá se přerušení od hodin v MCU, jejichž takt je 32768 Hz. Pro naše účely stačí, když jejich frekvence bude 32,768 Hz, což je 1 ms. V systému jsou také definovány 3 globální časovače. Volají se postupně a je dobré mezi nimi nechat alespoň 20 ms prodlevu pro vykonání instrukcí. Zvláště u periodického časovače. Kontrolují se tam všechna vytvořená spojení a pokud jich je víc, chvíli to trvá.

- periodic_timer – provádí pravidelnou kontrolu jednotlivých spojení, zda nemají data na odeslání.
- spinet_timer – kontroluje nová data v SPINETu
- fifo_timer – kontrola nových dat ve FIFO

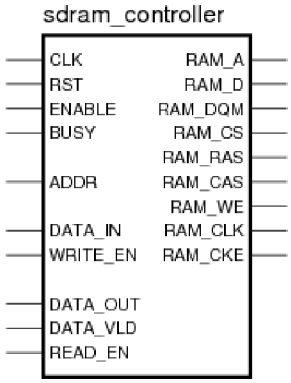
uIP používá pro každé spojení datovou strukturu obsahující mnoho údajů. Každé spojení se navazuje stejným způsobem podle stavového automatu pro sestavení, udržování a ukončení spojení.



Obr 6: Schéma činnosti TCP spojení[8]

I přes nízkou spotřebu RAM paměti však ani při minimální použitelné konfiguraci nebylo možné uIP ve stávající podobě využít. Důvodem byla mimo jiné knihovna libfitkit, která sama o sobě využívá zhruba polovinu dostupné RAM.

Proto bylo nutné využít paměť SDRAM na FITKitu, která je připojena přes rozhraní SPI k MCU a přesunout do ní větší část proměnných. Nejdříve bylo potřeba naprogramovat řadič SDRAM paměti. Vzhledem však k tomu, že nebyla potřeba žádná složitá funkce, použil jsem již hotový řadič z demo aplikace v SVN.



Obr 7: entita SDRAM řadiče[7]

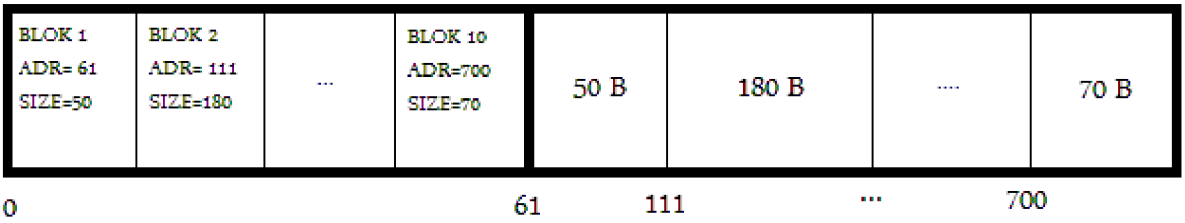
3.1.1 Knihovna SDRAM

Abychom mohli SDRAM paměť využít naplno, bylo nutné implementovat knihovnu funkcí pro práci s pamětí (jednoduchou správu). V této knihovně se nacházejí funkce pro načítání a ukládání dat do SDRAM paměti FITKitu.

Základem je paměťový blok, který obsahuje délku dat uložených v SDRAM a adresu těchto dat. V paměti SDRAM je poté její počátek (počet bloků * velikost bloku) rezervován právě pro tyto bloky a až poté následují vlastní data, na které odkazují adresy v jednotlivých blocích.

V lokální paměti RAM je definováno pole, kde jsou uchovávána čísla každého alokovaného bloku. Rozsah hodnot je 0 – maximální počet bloků, kde maximum je nejvyšší hodnota proměnné typu **char** (je možné použít také datový typ **short**, pro zvětšení počtu alokovaných položek), tedy 128 (32 768). Pokud blok není využitý, je v poli nastavena –1. Vzhledem k tomu, že pole je typu **char**, zabírá libovolně velký blok dat v SDRAM pouze 1B v lokální paměti (v případě použití short to budou 2B).

Velikost bloku je při standardním nastavení 6B (4B adresa, 2B velikost dat). Při kompilaci se musí nastavit bázová adresa SDRAM (standardně 0x00). Je také využívána globální proměnná, která ukládá offset od bázové adresy. Následující obrázek poskytuje ilustraci funkce.



Obr 8: Princip ukládání dat v SDRAM

Dvojitá adresace sice možná působí zbytečně, avšak finální úspora dat je značná.

Popis některých funkcí knihovny SDRAM:

- SDRAM_Init – Inicializace paměťových bloků v SDRAM
- SDRAM_Alloc – Alokuje nový blok dat v paměti
- SDRAM_WriteData – Zapíše data do SDRAM
- SDRAM_ReadData – Načítá data z SDRAM
- SDRAM_GetDataAddress – Vrací adresu dat uložených v SDRAM
- SDRAM_GetDataSize – Vrací velikost dat uložených v SDRAM
- SDRAM_Free – Uvolňuje paměťový blok pro další použití
- SDRAM_WriteVar – Zapisuje data do SDRAM za použití paměťového bloku
- SDRAM_ReadVar – Načítá data z paměťového bloku do lokálního bufferu
- SDRAM_CopyData – Kopíruje data z jedné adresy SDRAM na jinou

3.1.2 Knihovna FIFO

Celá FIFO fronta je uložena v paměti SDRAM a využívá funkce z knihovny SDRAM. V lokální paměti je použita struktura, která obsahuje informace o celkovém počtu uložených prvků, začátku a konci fronty. Oproti klasické frontě využívá ještě statické pole, které uchovává velikosti jednotlivých položek ve frontě, aby nedocházelo ke zbytečnému načítání velikosti bloku z SDRAM při každé operaci nad frontou. V souboru tcpip.h je nutné ještě nastavit parametry fronty:

- FIFO_BASE – může se nastavit přímo, avšak standardně fronta začíná za příchozími a odchozími buffery
- FIFO_LEN – velikost fronty. Zadává se počet paketů (veliksot paketu musí být nastavena)

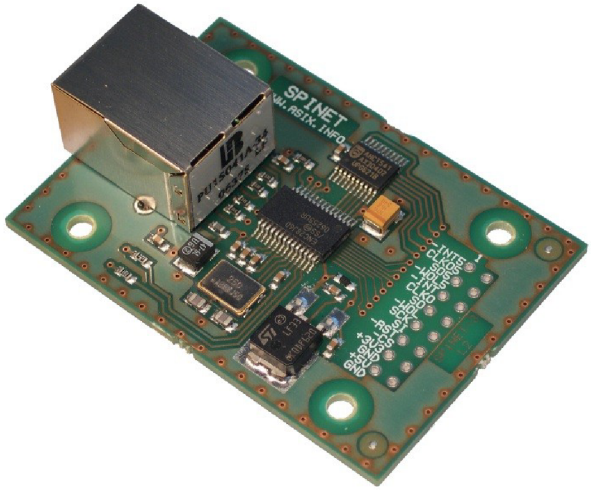
Popis některých funkcí knihovny FIFO:

- FIFO_Init – inicializace FIFO fronty
- FIFO_Add – přidá prvek do fronty z lokální paměti
- FIFO_AddPart – přidá pouze část prvku do fronty (vhodné při postupném generování dat, která patří k sobě
- FIFO_Read – načte položku z fronty do lokální paměti
- FIFO_ReadPart – načítá položku z fronty do lokální paměti po částech (vhodné při malém lokálním bufferu)
- FIFO_DropItem – Odstraní poslední položku z fronty (volá se typicky po přečtení paketu)
- FIFO_IsEmpty – test na prázdnost fronty
- FIFO_IsFull – test na plnost fronty
- FIFO_AddSDRAM – Přidá položku do fronty přímo z SDRAM

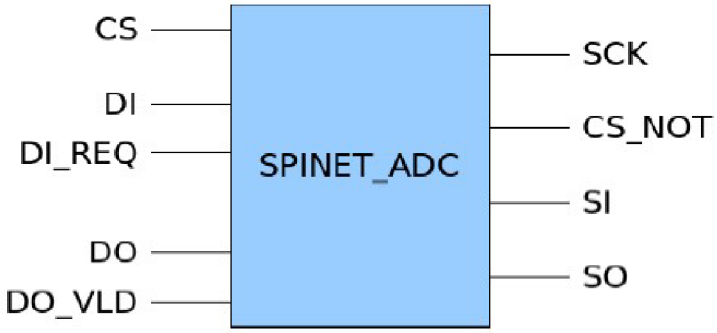
3.2 SPINET

Citace: „SPINET poskytuje fyzickou vrstvu 10BASE–T (PHY) včetně konektoru RJ45 a logiku Media Access Control (MAC). Obvod ENC28J60, na němž je modul SPINET založen, je samostatný Ethernetový řadič vyhovující normě IEEE 802.3. Podporuje plně duplexní i poloduplexní komunikaci rychlostí 10 Mb/s a je vybaven dvouportovou SRAM o velikosti 8 KB pro příchozí i odchozí data. Kromě jiných vlastností obsahuje i hardwarovou podporu pro kopírování bloků paměti a pro výpočet kontrolního součtu (IP checksum) a CRC. K externímu mikrokontroléru se připojuje prostřednictvím rozhraní SPI, které může být snadno realizováno jakýmkoli mikrokontrolérem a vyžaduje pouze 4 nebo 5 signálových vodičů.“ [5]

V našem případě se SPINET používá pouze pro připojení FITKitu k síti. Žádné rozšíření v podobě výpočtu CRC nebo checksum hlaviček se nevyžívá, stejně tak není využito ani nastavování MAC adresy. Vše se provádí softwarově v MCU a do spinetu je nahrán již hotový paket. Původně byl totiž záměr otestovat uIP samo na sobě a tomu byla uzpůsobena koncepce. Bylo by však zapotřebí naprogramovat i server a to by bylo náročné na aplikaci i zdroje. SPINET mi byl doporučen až později.



Pro zprovoznění SPInetu je potřeba nahrát do FPGA entitu spinetu. Zde jsem si dovolil opět využít bakalářskou práci Bc. René Kolaříka a použít jeho VHDL design spolu s výše uvedenou SDRAM entitou.



Obr 10: Entita VHDL[6]

3.2.1 Knihovna ENC28J60

Tato knihovna obsahuje funkce a nastavení pro inicializaci spinetu, přijímání a odesílání paketů. Je volně dostupná na stránkách spinetu jako demonstrační aplikace. Z nich se využívají zejména:

- netbuf_write – zápis dat do paměti spinetu
- netbuf_read – čtení dat
- nic_transalloc – zjistí kolik místa je k dispozici pro zápis dat a zapíše kontrolní byte
- nic_transmit – spustí přenos dat
- nic_receive – zjišťuje příchod nových dat do spinetu
- nic_recfree – uvolní paměť pro další použití (posune ukazatel za zpracovávaný paket)

4 Implementace

Pro implementaci celého systému bylo nutné vytvořit vhdl entitu obsahující jak řadič SDRAM, tak řadič spinetu. Při následném pokusu o spuštění však vyvstaly závažné problémy znemožňující použití FIFO a SDRAM knihoven. SPINET totiž vyžaduje ke své funkčnosti takt SPI rozhraní 8 – 20 MHz. Řadič RAM paměti však vyžaduje takt minimálně 50 MHz, v novější verzi řadiče pak 100 MHz. I přes různé způsoby zapojení a pokusů se mi ani za použití DCM[2] entit a nastavování taktu SPI rozhraní v FPGA nepodařilo zprovoznit obě zařízení současně. Buď fungoval spinet, nebo řadič SDRAM. Vyvstal mi tedy problém: Buď budu mít TCP/IP protokol využívající FIFO frontu a externí SDRAM, ale neotestuji ho, nebo mít ořezanou verzi TCP/IP využívající pouze lokální buffery, ale s spinetem, pomocí kterého výsledek otestuji. Zvolil jsem si druhou možnost. Získá se tím alespoň omezená funkčnost a veškerá rozšíření jsou k dispozici do budoucna.

Program byl tedy upraven tak, aby výsledná podpora SDRAM a FIFO knihovny byla co nejjednodušeji použitelná v případě, že bude vyřešen problém FITKitu s použitím SDRAM paměti. V hlavičkovém souboru tcpip.h existuje definice SDRAM, jejímž nastavením na hodnotu 0 říkáme, že podpora SDRAM je vypnuta. Naopak nastavením na 1 ji povolíme. Vzhledem však k tomu, že celý systém není dosud možné otestovat s využitím fronty, je použití čistě experimentální a nezaručuje požadovanou cílovou funkčnost. Ve výsledném programu je tedy podpora SDRAM vypnuta.

Dále zde existují 2 časovače, které zajišťují cyklickou kontrolu systému. Periody obou jsou definovatelné v souboru clock.h. Prvním z nich je PERIODIC_TIMER. Ten určuje, po jaké době se má provádět cyklická kontrola všech spojení. V podobě, v jaké je program teď, by bylo možné periodickou kontrolu vypnout, protože bez podpory SDRAM je počet připojení omezen jen na jedno. Druhý časovač – TIMER_SPINET_CHECK – určuje periodu zjišťování nových dat v spinetu. Při každém taktu se dotazuje, zda v spinetu nejsou data (za použití funkcí z knihovny enc28j60). Pokud by však byl k dispozici vestavěný ethernetový kontrolér, můžeme s výhodou použít TIMER_BUFFER_CHECK. Tím by se zapnula možnost kontrolovat data přímo v SDRAM za pomocí příznaků, které jsou uloženy na adrese SDRAM_PHYS_FLAGS_ADDR (adresa před začátkem FIFO fronty). Předchystány jsou příznaky:

2 Digital Clock Management – zařízení v FPGA umožňující generovat libovolné frekvence hodin za použití násobení a dělení původního přivedeného hodinového signálu. Dokáže také libovolné fázové posuny a další.

- FLAG_NONE – žádná operace se nepožaduje
- FLAG_OUTGOING_PACKET_READY – data připravená na odeslání
- FLAG_INCOMING_PACKET_READY – přijatá nová data
- FLAG_TRANSMIT_OUT – přenášení odchozích dat
- FLAG_TRANSMIT_IN – přenášení příchozích dat

Je však nutná HW podpora ze strany budoucího ethernetového kontroléru. Ten musí umět nastavovat tyto příznaky na zadané adrese, aby mohly být programem zpracovány. Doprogramování obsluhy příznaků v programu je již poté velmi jednoduché. Při každé periodě časovače lze použít stavový automat, kde každý stav je daný daný příznak. Pak lze volat funkce z knihovny FIFO pro ukládání a načítání dat.

Výsledná funkčnost programu tedy vypadá následovně: v nekonečné smyčce se volají pomocí přerušení hodin 2 výše zmíněné časovače. Při výskytu dat v spinetu se nahrají do globálního bufferu. Poté se volá procesní funkce TCP/IP, která nastaví příslušné události a zavolá obslužnou funkci. Procesní funkce zkontroluje, o jaký typ dat se jedná (TCP, UDP, ICMP, ...) a podle toho provrde obsluhu.

Pokud má aplikace data k odeslání, volá se v procesní funkci funkce TCPIP_Send. Uvnitř ní se volá funkce TCPIP_SetLinkLayer, která vyhledá v ARP tabulce cílovou IP a pokud existuje, zajistí nastavení cílové a zdrojové MAC adresy a také typu protokolu. Pokud záznam nenalezne, zajistí vyslání ARP paketu s požadavkem na cílovou adresu.

Díky omezenému paměťovému prostoru MCU a nemožnosti jeho rozšíření v SDRAM muselo být nastaveno omezení na jediné TCP spojení. Podpora UDP byla implementována pouze na příchozí data, ale je 100% připravena i na odchozí data. ICMP podpora je pouze na příkaz ping (echo reply). Zpracovává se stejně jako jiné pakety, tedy pomocí uIP a stavového automatu. Maximální velikost datového bufferu byla nastavena na 256B. Stejná je tedy i hodnota MSS[3]. Je to rozumný kompromis mezi paměťovou náročností a počtem příchozích paketů. Musela být také vypnuta fragmentace, protože díky ní by byl potřeba další buffer stejné velikosti. Lze jej však využít za pomoci uIP, přesunout jej do SDRAM a využít stejné funkce pro práci s SDRAM jako v případě příchodu běžných dat (ne jen fragmentu)

Všechna tato omezení by padla v případě využití externí paměti. Interní buffer by byl sice stále stejné velikosti, avšak navenek bychom mohli načítat pakety libovolné velikosti, ty ukládat do FIFO a následně po částech zpracovávat v MCU.

3 MSS – maximum segment size – maximální celková velikost příchozího paketu včetně hlaviček

4.1 Knihovna DEBUG

V průběhu vývoje vyvstala potřeba kontroly chování programu. FITKit bohužel nemá žádné debugovací nástroje, a tak musela být vytvořena alespoň tato knihovna. Debugování se v tomto případě zakládá pouze na testovacích výpisech při průchodu jednotlivými větvemi programu. Téměř každá podmínka obsahuje text s jejím názvem, aby bylo při činnosti poznat, kde se právě nalézáme. Možnost debug módu lze přidat nastavením definice DEBUG na 1 v souboru tcpip.h. Standardně je knihovna vypnuta – tedy nastavena na 0.

Knihovna ale také obsahuje funkce umožňující kontrolní výpisy lokální paměti, SDRAM paměti, FIFO fronty, stavu spojení a také výpis přijatých paketů. Veškeré funkce začínají prefixem DEBUG.

4.2 Ukázkový program

Pro demonstraci funkčnosti jsem vytvořil jednoduchý HTTP server. Vzhledem k tomu, že jsme značně limitování jak RAM pamětí, tak FLASH pamětí pro program, musel jsem veškeré stránky vytvořit jako statické definice v programu, které mají jen jednoduchou formu.

Internetový prohlížeč vysílá standardně na port 80. Ten tedy v aplikaci naslouchá. Pokud zadáme jen adresu FITKitu, ten vrátí stránku index. Dále obsahuje stránku stat.html, která vypíše statistiku příchozích paketů (jejich typů, počet zahozených, apod.) Pokud zadáme jinou stránku, kterou FITKit nezná, vrátí stránku 404 (nenalezeno), v případě použití jiného protokolu než HTTP 1.0 nebo HTTP 1.1 vrátí stránku 501 (služba nedostupná). Jedná se pouze o jednoduché výpisy do internetového prohlížeče. Demostrační program nalezneme na přiloženém CD v adresáři „tcpip“.

Po spuštění programu lze využít terminál pro základní nastaveni a obsluhu. Lze zadat následující parametry:

- setip ip – nastaví IP FITKitu. Ta může být jakákoliv IPv4 adresa
- listen port – nastaví naslouchání na portu zadaném v parametru. Port může být jakýkoliv v rozmezí 1 – 65535
- unlisten port – zruší naslouchání na zadaném portu.
- showarp – zobrazí ARP tabulku
- showconns – zobrazí aktuální seznam připojení

Vzhledem k absenci pokročilejšího komunikačního programu pro fitkit nemá smysl sestavovat aktivní spojení ze strany FITKitu. Je však na toto plně připraven.

Před samotným spouštěním je vhodné si přečíst soubor „readme“ na CD. Obsahuje informace potřebné ke zprovoznění tohoto programu.

5 Závěr

Cílem bylo navrhnout a implementovat TCP/IP protokol, který by splnil nároky vestavěného systému. Jako vhodné řešení byl vybrán TCP/IP „stack“ uIP. Ten tyto nároky splňuje. Zásadním problémem bylo otestování výsledného systému. K tomu je využito zařízení spinet od firmy ASIX. O tom pojednával ve své práci Bc. René Kolařík. S využitím jeho poznatků a VHDL designu se podařilo TCP/IP otestovat.

V původním konceptu měl program využívat externí paměť SDRAM, ve které měla být implementována FIFO fronta. Ta měla být určena k průběžnému ukládání příchozích paketů a jejich pozdějšímu zpracování v MCU. Bohužel se však nepodařilo současně zprovoznit spinet a řadič SDRAM. Důvodem toho byla rozdílná rychlost SPI rozhraní požadovaná řadiči. Ani po několika pokusech se nepodařilo oba řadiče zprovoznit.

Nejdříve jsem vyzkoušel použít dvoje různé hodiny – pro řadič SDRAM hodiny SMCLK z MCU (7,92 MHz) a pro spinet hodiny ACLK, které byly násobeny v DCM na požadovanou hodnotu. Toto nevedlo k výsledku. Poté mi bylo doporučeno použít stejné hodiny SMCLK, násobené v DCM na požadovanou hodnotu SDRAM řadičem, avšak jako takt spinetu byl využít výstup násobený jedničkou – tedy stejný signál SMCLK. Změna bázových adres spolu s kombinací postupů výše a dostatečný vzájemný odstup adres od zařízení rovněž nebyl úspěšný. Vždy fungovalo jen jedno zařízení – pokud byl do proměnné SPI_CLK v top level entitě puštěn signál SMCLK, fungoval spinet, pokud 50 MHz z DCM jednotky, pak fungovala SDRAM paměť.

Cílový systém je tedy ochuzený o původně plánované vlastnosti, avšak po úspěšném zprovoznění obou řadičů je připravený na plný provoz pouhou změnou několika parametrů a drobnými úpravami kódu. Dále je systém připraven pro budoucí vestavěný ethernetový kontrolér, který by mohl FITKit přímo obsahovat.

Další rozšíření se přímo nabízí. Implementovat protokoly vyšších vrstev a využít FITKit jako webový, DNS nebo DHCP server.

Další varianta by pak byla vytvořit ethernetový kontrolér přímo do FITKitu, který by zpracovával příchozí pakety z SDRAM. Ušetřil by se tím neustálý přenos z spinetu do SDRAM a posléze do MCU. Drobnou nevýhodou tohoto řešení je zabrání místa v FPGA. Pokud by však FITKit sloužil čistě jako síťové zařízení, nebylo by to na závadu.

Literatura

[1] InetDeamon – ARP, www stránky (1/2009)
http://www.inetdaemon.com/tutorials/networking/lan/arp.shtml

[2] InetDeamon – IPv4, www stránky (1/2009)
http://www.inetdaemon.com/tutorials/internet/ip/datagrams.shtml

[3] Wikipedie – TCP, www stránky elektronické encyklopedie (1/2009)
http://www.inetdaemon.com/tutorials/internet/tcp/tcp_header.shtml

[4] Dunkels Adam and the Swedish Institute of Computer Science: uIP, Sweden, 2001–2003
http://www.sics.se/~adam/uip/index.php/Main_Page

[5] Zařízení Spinet firmy ASIX – www stránky (1/2009)
http://www.asix.cz/a6_spinet.htm

[6] René Kolařík: Rozhraní ethernet pro výukovou HW/SW platformu FITKit, bakalářská práce, Brno, FIT VUT v Brně, 2008

[7] FITKit, informace o SDRAM a SPI – www stránky (1/2009)
http://merlin.fit.vutbr.cz/FITkit/docs/firmware/sdramctrl.html

[8] TCP/IP Guide, internetová příručka
http://www.tcpipguide.com/free/t_TCPOperationalOverviewandtheTCPFiniteStateMachineF-2.htm

Seznam příloh

Příloha 1. CD se zdrojovými texty