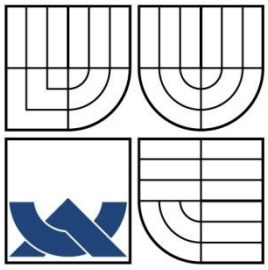
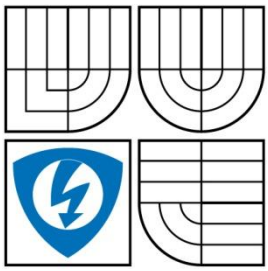


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF

IMPLEMENTACE JEDNODUCHÉHO WEB SERVERU DO MIKROKONTROLÉRU COLDFIRE MCF 52233

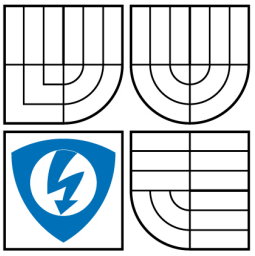
TITLE

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — BC. DAVID KOTÍK
AUTHOR

VEDOUCÍ PRÁCE — ING. TOMÁŠ MACHO, PH.D.
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. David Kotík	***ID:***	112289
Ročník:	2	***Akademický rok:***	2008/2009

NÁZEV TÉMATU:

Implementace jednoduchého web serveru do mikrokontroléru ColdFire MCF 52233

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s mikrokontroléry řady ColdFire MCF 5223x firmy FreeScale a s Free ColdFire TCP/IP stack by Interniche.
2. Prostudujte protokol HTTP 1.1.
3. Navrhněte a implementujte jednoduchý web server do mikrokontroléru MCF 52233. Web server by měl dovolovat vyčítání hodnot binárních vstupů, výstupů z A/D převodníků a zapisovat hodnoty do binárních výstupů.
4. Ověřte funkčnost web serveru.

DOPORUČENÁ LITERATURA:

MCF52235 ColdFire Integrated Microcontroller Reference Manual rev. 5. FreeScale semiconductor, 2007.
Dále dle vlastního výběru.

Termín zadání:	9.2.2009	***Termín odevzdání:***	25.5.2009

Vedoucí práce: Ing. Tomáš Macho, Ph.D.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt
Cílem této diplomové práce je návrh a implementace webového serveru do mikrokontroléru ColdFire MCF 52233. V diplomové práci je začátek věnován samotnému mikrokontroléru MCF 52233 a jeho struktuře. Dalšími částmi jsou přehled komunikace pomocí protokolu http a ColdFire TCP/IP stack, včetně seznamu API funkcí stacku. Pro větší přiblížení práce se stackem je uveden jednoduchý příklad implementace API funkcí. Vlastní program je potom postupně popisován, podle způsobu jakým pracuje.

Klíčová slova
ColdFire MCF 52233, Free ColdFire TCP/IP od Interniche, HTTP 1.1

Abstract
The goal of this master's thesis is design and simple web server implementation into microcontroller ColdFire MCF 52233. Begin of the master's thesis is about microcontroller MCF 52233 and his architecture. Next parts represent overview of communication protocol http and ColdFire TCP/IP stack inclusive list of API functions of stack. For better understanding how to work with ColdFire TCP/IP stack is introduced the simple example of implementation API functions. Characteristics of the program are described step by step according to way of working.

Keywords
ColdFire MCF 52233, Free ColdFire TCP/IP by Interniche, HTTP 1.1

KOTÍK, D. *Implementace jednoduchého web serveru do mikrokontroléru ColdFire MCF 52233*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. XY s. Vedoucí diplomové práce Ing. Tomáš Macho, Ph.D.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Implementace jednoduchého web serveru do mikrokontroléru ColdFire MCF 52233 jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne: **25. května 2009** ……

podpis autora

Pod□kování

Děkuji vedoucímu diplomové práce Ing. Tomáš Machovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **25. května 2009** ……

podpis autora

OBSAH

1. ÚVOD

Pokud bude naším požadavkem přenášet data ze senzoru na větší vzdálenost, nebo na větší vzdálenost ovládat nějaké spotřebiče, máme na výběr veliký počet standardů pro komunikaci. Vývěr provádíme podle prostředí, ve kterém zařízení bude komunikovat, potřebné rychlosti komunikace, vzdálenosti, ceny zařízení a v neposlední řadě také zabezpečení. U každé metody komunikace je zapotřebí médium, po kterém bude komunikace probíhat. Pokud opomineme bezdrátovou komunikaci, budou signály přenášeny za pomocí kabelů. U různých standardů nebývá většinou možné používat společné přenosové médium a je tedy potřeba při realizaci spojení zavádět kabely nové. V dnešní době se staly ethernetové sítě LAN standardem používaným ke spojení osobních počítačů v rámci lokální sítě.
S takovouto sítí se běžně setkáváme nejenom u firem, ale také v domácnostech. A tak je po stránce řešení síťových rozvodů velice výhodné vytvořit zařízení schopné pracovat v síti ethernet.
Tématem diplomové práce je implementace jednoduchého webového serveru do mikrokontroléru ColdFire MCF 52233. Jehož bezespornou výhodou je, že můžeme po síti ethernet přenášet data ze senzorů a ovládat výstupy. A přitom nebude nutné žádné další softwarové vybavení osobního počítače k práci s takovýmto zařízením, protože jej budeme ovládat za pomocí webového prohlížeče. Dokonce připojením sítě LAN k internetu získáme možnost pracovat se zařízením na libovolnou vzdálenost.

2. MIKROKONTROLÉRY COLDFIRE ŘADY MFC 5223X FIRMY FREESCALE

Hovorově řečeno „vlajkovou lodí“ mikrokontrolérů MFC 5223x je mikrokontrolér MFC52235 obsahující veškeré vnitřní periférie dostupné pro tuto řadu. Proto se zaměříme na jeho popis a pro porovnání ostatních modifikací tohoto mikrokontroléru naleznete přehlednou tabulku č.1 na konci této kapitoly.

2.1 KRÁTKÝ PŘEHLED

Mikrokontrolér MCF52235 a jeho modifikace jsou obvody s vysokým stupněm integrace celé rodiny mikrokontrolérů s obchodní značkou ColdFire se zmenšenou instrukční programovací sadou (RICS) obsahující typy MCF52230, MCF52231, MCF52232, MC52233, MC52234 a MCF52236.
MCF52235 je jako hlavní zástupce vysoce integrovaných 32-bitových mikrokontrolérů založen na mikroarchitektuře ColdFire V2. Vlastnostmi, jako jsou až 32 Kbytů interní paměti SRAM a až 256 Kbytů flash paměti, čtyři 32-bitové časovače s možností požadavku DMA, čtyř kanálovým DMA kontrolérem, rychlým Ethernet kontrolérem, CAN modulem, I^2C^{TM} modulem, třemi sériovými rozhraními UART a řadou rozhraní SPI, nachází obvod MC52235 snadno své uplatnění v průmyslových aplikacích.
Toto 32-bitové zařízení založené na verzi jádra ColdFire číslo dvě (V2), se sníženou instrukční programovací sadou s rozšířeným násobícím akumulátorem (EMAC).
Následující moduly jsou obsaženy na čipu:

- V2 ColdFire jádro s rozšířeným akumulátorem pro násobení (EMAC)
- Kryptologická urychlovací jednotka (CAU)
- Až 32 Kbytů interní paměti SRAM
- Až 256 Kbytů interní flash paměti
- Fast Ethernet kontrolér (FEC) s přijímačem na čipu (ePHY)
- Tři univerzální asynchronní přijímače/vysílače (UART)
- Modul kontroléru area network 2.0B (FlexCAN)

- Vnitřní integrovaný obvod sběrnicového kontroléru (I^2C)
- Deseti nebo dvanácti bitový analogově-digitální převodník (ADC)
- Moduly pro řazené rozhraní sériových periférií (QSPI)
- Čtyř kanálový, 32-bitový kontrolér přímého přístupu do paměti (DMA)
- Čtyř kanálové, 32-bitové vstupně zachytávací/výstupně porovnávací časovače s volitelnou podporou DMA
- Dva 16-bitové periodicky přerušované časovače (PIT)
- Programovatelný softwarový watchdog časovač
- Dva kontroléry přerušení, každý schopný zvládnout až 63 zdrojů přerušení (126 celkem)

Tyto součástky jsou ideální pro cenově citlivé aplikace vyžadující v hlavní řadě schopnosti připojení. Dále výborně poslouží jako oddělovače dat a různá uživatelská rozhraní, nebo zpracování signálů v různých klíčových oblastech trhu, mezi něž patří zabezpečovací technika, zpracování obrazů, komunikační sítě, hry a medicínské vybavení. Toto spojení množství integrovaných periférií a vysokého výkonu, umožňuje společně se snadným znovu použitím kódu a rozsáhlou podporou nástrojů výrobců třetích stran, v krátké době uvádět nové výrobky na trh.

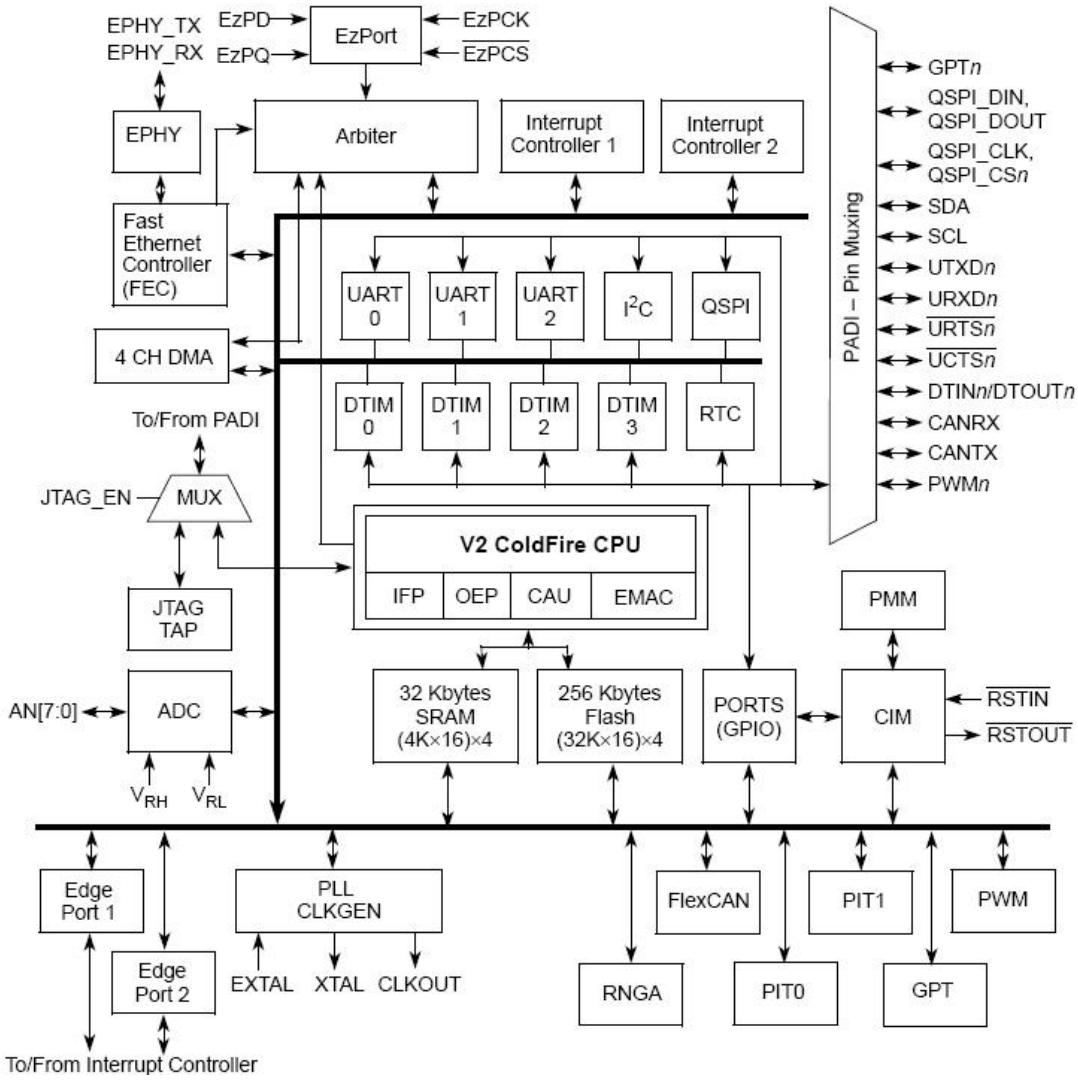
2.2 VLASTNOSTI MIKROKONTROLÉRŮ ŘADY MCF52235

Blokové schéma mikrokontroléru zobrazeného na obrázku č. 2.1 je dále popsáno podrobnějším přehledem jeho vlastností.

- Verze 2 ColdFire s proměnnou délkou RISC jádra procesoru
  - statické operace
  - 32- bitová datová a adresová sběrnice na čipu
  - frekvence jádra až 60 MHz
  - šestnáct volných 32-bitových datových a adresových registrů

  - implementované ColdFire ISA_A s rozšířenou podporou pro uživatelem definovaný registr ukazatel zásobníku a čtyři nové instrukce pro zlepšení bitových operací (ISA_A+)
  - rozšířený akumulátor pro operace násobení (EMAC) s 32 bitovým akumulátorem pro podporu násobení 16x16 -> 32 nebo 32x32 -> 32
  - kryptovací akcelerační jednotka (CAU)
  - úzce spojený koprocesor pro zrychlení softwarově založeného krytování a funkcí přehledu zpráv
  - FIPS – 140 generátor náhodných čísel
  - podpora algoritmů DES, 3DES, AES, MD5 a SHA-1
  - detekce neplatných instrukcí, umožňující 64k emulaci
- Podpora odlaďování v systému
  - real-timové sledování pro detekci směru spouštěného procesu
  - mód odlaďování programu na pozadí (BDM) pro odlaďování přímo v obvodu (DEBUG_B+)
  - real-timová podpora odlaďování se šesti hardwarovými breakpointy (4 PC, 1 adresový a 1 datový), které mohou být konfigurovány na 1 nebo 2 úrovňové spouštění
- Paměť na čipu
  - až 32 kbytů dvoukanálové paměti SRAM na interní sběrnici v jádře CPU a DMA přístup s podporou standby módu
  - až 256 kbytů prokládané Flash paměti s přístupem 2-1-1-1
- Správa napájení
  - plně statické operace se spícím procesorem v době nečinnosti a stop mód pro celý čip
  - rychlá odezva na přerušení z nízko příkonového spícího módu (wake-up feature)
  - zakázání/povolení hodinových impulzů pro každou periférii v případě jejího nepoužívání nebo potřeby

Obr. 1. *Blokové schéma mikrokontroléru MCF52235*



- Fast Ethernet kontrolér (FEC)
  - 10/100 BaseT/TX, poloviční nebo plný duplex
  - přijímací / vysílací paměť typu FIFO na čipu
  - vnitřní DMA kontrolér
- Ethernetový přijímač na čipu (EPHY)
  - digitální adaptivní egalizace
  - podpora auto negace
  - základní korekce oběžníků

  - plný / poloviční duplex u všech modelů
  - módy se zpětnou smyčkou
  - podpora MDIO úvodního potlačování
  - JUMBO pakety
- FlexCAN 2.0B modul
  - obsahuje všechny existující vlastnosti Freescale TouCAN modulu, na kterém je založen
  - plná implementace CAN protokolu specifikovaného verzí 2.0B
  - standardní datový a vzdálený rámec (až 109 bitů dlouhý)
  - rozšířený datový a vzdálený rámec (až 127 bitů dlouhý)
  - 0 až 8 bytová délka dat
  - programovatelná bitová přenosová rychlost až 1 Mbit / sek
  - pružný buffer zpráv (MBs), celkově až 16 bufferů pro zprávy délky 0 až 8 bytů, konfigurovatelný jako Rx nebo Tx, všechny podporují standardní a rozšířené zprávy
  - nepoužitý buffer zpráv může být použit jako RAM pro všeobecné využití
  - mód naslouchání oprávnění
  - obsahově spojené adresování
  - není potřeba čtecí / zapisovací signalizace
  - tři programovatelné masky registrů: komplexní pro MB 0 – 13, speciální pro MB 14 a speciální pro MB 15
  - programovatelný plán prvního vysílání: nejnižší ID nebo nejnižší číslo bufferu
  - časová známka založená na 16-bitovém volně běžícím časovači
  - globální síťový čas, synchronizovaný specifickou zprávou
  - maskované přerušení

- Tři univerzální asynchronní / synchronní přijímače a vysílače (UART)
  - 16-bitová dělička pro generování hodinového signálu
  - Kontrolní logika přerušení s maskovatelným přerušením
  - DMA podpora
  - Datový formát může být 5, 6, 7 nebo 8 bitů se sudou, lichou nebo žádnou paritou
  - až dva stop bity v 1 / 16 inkrementu
  - schopnost chybové detekce
  - podpora modemu obsahující „žádost o poslání“ (RTS) a „čisté pro poslání“ (CTS) vývody pro dva UART porty
  - vysílací a přijímací FIFO buffery
- I^2C modul
  - uvnitř čipu obsažená sběrnice pro připojení EEPROM, LCD kontroléry, A / D převodníky a klávesnice
  - plně kompatibilní s průmyslovým standardem I^2C sběrnice
  - master a slave módy podporující násobného mastera
  - automatické generování přerušení s programovatelnou úrovní
- Řada sériových vnějších rozhraní (QSPI)
  - plně duplexní, tří vodičovi synchronní přenos
  - dostupný výběr až čtyřech čipů
  - mód operování pouze jako master
  - programovatelná bitová rychlost až na polovinu hodinové frekvence procesoru
  - až 16 předprogramovaných přenosů
- Rychlý analogově digitální převodník (ADC)
  - osm analogových vstupních kanálů
  - dvanácti bitové rozlišení
  - minimální čas převodu 1,125 μs
  - současné vzorkování dvou kanálů pro aplikace řízení motorů
  - jednoprůchodové nebo kontinuální operace

  - volitelné přerušení při kompletním převodu, nulovém průchodu (změna znaménka), nebo limitů pod / přes a nízký / vysoký
  - nepoužitý analogový vstup může být použit jako digitální vstup / výstup
- Čtyři 32-bitové DMA časovače
  - rozlišení 17 ns při 60 MHz
  - programovatelné zdroje pro vstup hodin, včetně možnosti volby pro externí časování
  - programovatelná dělička
  - možnost módu snímání s programovatelnou hranou spouštění na vstupním pinu
  - porovnávání výstupu s programovatelným módem pro výstupní pin
  - módy volného běhu a restartu
  - maskované přerušení podle snímaného vstupu nebo porovnáním výstupu
  - možnost spouštění DMA na základě snímaného vstupu nebo porovnáním výstupu
- Čtyř kanálové časovače pro všeobecné použití
  - 16-bitová architektura
  - programovatelná dělička
  - proměnná šířka výstupního impulzu od mikrosekund až do sekund
  - samotný 16-bitový vstupní pulzní akumulátor
  - toggle-on-overflow označovaná vlastnost pro generování pulzně šířkové modulace (PWM)
  - jeden dvou módový pulzní akumulační kanál

- Pulzně šířkový modulační časovač
  - může pracovat jako osmikanálový s 8-bitovým rozlišením nebo čtyř kanálový s šestnácti bitovým rozlišením
  - programovatelná perioda a šířka impulzů
  - programové povolení / zákaz funkce pro každý kanál
  - softwarově vybíraná polarita pro každý kanál
  - perioda a šířka impulzu mají dvojitou vyrovnávací paměť, změna se projeví, když je dosaženo konce současné periody (PWM čítač dosáhne nuly) nebo když je kanál zakázán
  - programovatelné středové nebo levé seřazení výstupů na individuálním kanálu
  - čtyři hodinové zdroje (A, B, SA a SB) poskytující široký rozsah frekvencí
  - pohotovostní vypnutí
- Hodiny reálného času (RTC)
  - udržovaný systémový denní čas
  - provádí funkce přerušení „stopky“ a „alarm“
- Dva periodické časovače přerušení (PIT)
  - 16-bitový čítač
  - možnost výběru jako volně běžící nebo odpočítávání
- Softwarový watchdog časovač
  - 32-bitový čítač
  - podpora nízko příkonového napájení
- Vlastnosti hodinového generátoru
  - 25 MHz vstup pro krystal
  - fázový závěs (PLL) implementovaný na čipu může generovat frekvence až do maximální frekvence procesoru 60 MHz
  - poskytuje hodinové impulzy pro integrované EPHY

- Dvojitý kontrolér přerušení
  - podpora pro zdroje násobného přerušení seřazené podle následujících zdrojů přerušení:
  - plně programovatelný zdroj přerušení pro každou periférii
  - sedm zdrojů přerušení s pevnou úrovní
  - sedm externích přerušovacích signálů
  - unikátní číslo vektoru pro každý zdroj přerušení
  - možnost maskování jakékoliv individuální zdroj přerušení nebo všechny zdroje přerušení (global mask-all)
  - podpora pro hardwarové a softwarové potvrzení cyklů přerušení (IACK)
  - kombinace cest pro umožnění probuzení z nízko příkonových módů
- DMA kontrolér
  - čtyři plně programovatelné kanály
  - podpora duálně adresovaného přenosu 8, 16 a 32-bitového, stálého s podporou pro 16-bytové (4 x 32-bitové) přenosy
  - zdrojové / cílové adresní ukazatele, které mohou být inkrementovány nebo mohou zůstávat konstantní
  - 24-bitový přenosový čítač na kanál
  - automatické zarovnávání přenosů pro efektivní blokové přesuny
  - podpora pro „překročení“ a „odcizení“ cyklu
  - softwarově programovatelný DMA požadavek pro UARTy (3) a 32-bitové časovače (4)
- Reset
  - oddělitelné vstupní a výstupní RESET signály
  - sedm zdrojů resetu
  - power-on reset (POR)
  - externí

  - softwarový
  - watchdog
  - ztráta hodinového signálu
  - ztráta uzamčení
  - detekce nízkého napětí (LVD)
  - příznaková indikace posledního zdroje resetu
- Integrovaný čipový modul
  - systémová konfigurace během resetu
  - výběr jednoho ze tří hodinových módů
  - konfigurace intenzity řízení výstupního bloku
  - jedinečné identifikační číslo součástky a kontrolní číslo součástky
- Vstupně / výstupní rozhraní pro všeobecné použití
  - až 56 bitů pro všeobecně použitelné I/O
  - bitová manipulace podporovaná pomocí set / clear funkcí
  - programovatelná velikost řízení
  - nepoužité piny periférií mohou být extra použity jako GPIO
- JTAG podpora pro testování na úrovni hardwaru

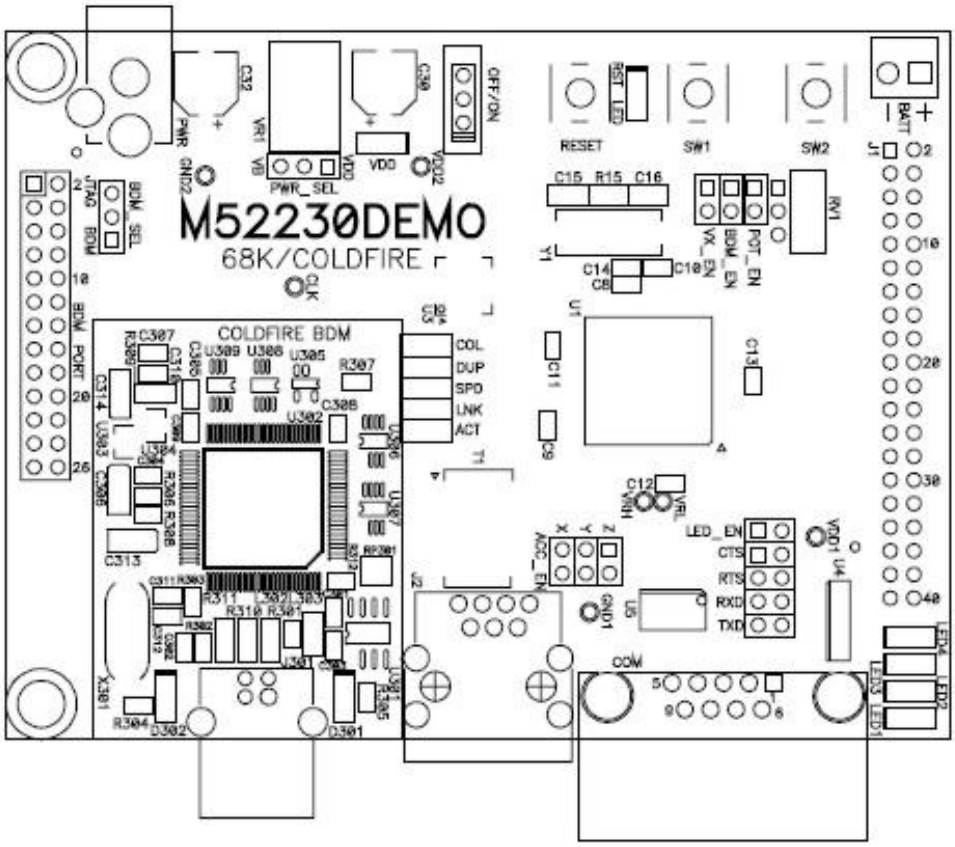
2.3 MOŽNOSTI IMPLEMENTACE JEDNODUCHÉHO WEB SERVERU DO MIKROKONTROLÉRU MCF 52233

Pro další praktickou práci a pro odzkoušení celého systému jsem si zvolil demonstrační kit od firmy FreeScale s názvem M52233DEMO (obr. 2). Součástí tohoto kitu je i vývojové prostředí s názvem CodeWarrior. Vývojový kit je založen na mikrořadiči MCF52233 v osmdesáti pinovém provedení. Obsahuje veškeré potřebné podpůrné obvody pro jeho práci a obsahuje i modul ladění programu na pozadí (BDM), což je velmi výkonný a užitečný prostředek pro práci s tímto mikrokontrolérem. K dispozici dále máme dva porty SCI, jeden QSPI, jeden IIC, dva RS-232 a ethernetové komunikační rozhraní s konektorem RJ-45, kompatibilní

s IEEE 802.3 na 10/100 Mbps. Celou desku výborně doplňují součástky, jako jsou tříosý akcelerometr, čtyři LED diody, dva spínače a trimr. Hodně aplikací tedy můžeme vyzkoušet jenom za pomoci tohoto kitu a nemusíme připojovat žádné externí součástky. I když pro tento účel deska obsahuje 40 pinový konektor na připojení rozšiřujících zařízení.

Cena tohoto kitu je v České Republice cca 3000 Kč, ve Spojených Státech 99$.

Obr. 2. *Provedení vývojového kitu M52230DEMO*



Tab. 1. *Rodina mikrokontrolérů řady MCF52235*

Module	52230	52231	52232	52233	52234	52235	52236
Version 2 ColdFire Core with EMAC (Enhanced Multiply-Accumulate Unit)	•	•	•	•	•	•	•
System Clock (MHz)	60	60	50	60	60	60	50
Performance (Dhrystone 2.1 MIPS)	56	56	46	56	56	56	46
Flash / Static RAM (SRAM)	128/32 Kbytes	128/32 Kbytes	128/32 Kbytes	256/32 Kbytes	256/32 Kbytes	256/32 Kbytes	256/32 Kbytes
Interrupt Controllers (INTC0/INTC1)	•	•	•	•	•	•	•
Fast Analog-to-Digital Converter (ADC)	•	•	•	•	•	•	•
Random Number Generator and Crypto Acceleration Unit (CAU)	—	—	—	—	—	•	—
FlexCAN 2.0B Module	—	•	—	—	•	•	—
Fast Ethernet Controller (FEC) with on-chip interface (EPHY)	•	•	•	•	•	•	•
Four-channel Direct-Memory Access (DMA)	•	•	•	•	•	•	•
Software Watchdog Timer (WDT)	•	•	•	•	•	•	•
Programmable Interrupt Timer	2	2	2	2	2	2	2
Four-Channel General Purpose Timer	•	•	•	•	•	•	•
32-bit DMA Timers	4	4	4	4	4	4	4
QSPI	•	•	•	•	•	•	•
UART(s)	3	3	3	3	3	3	3
I^2C	•	•	•	•	•	•	•
Eight/Four-channel 8/16-bit PWM Timer	•	•	•	•	•	•	•
General Purpose I/O Module (GPIO)	•	•	•	•	•	•	•
Chip Configuration and Reset Controller Module	•	•	•	•	•	•	•
Background Debug Mode (BDM)	•	•	•	•	•	•	•
JTAG - IEEE 1149.1 Test Access Port[1]	•	•	•	•	•	•	•
Package	80 LQFP 112 LQFP	80 LQFP 112 LQFP	80 LQFP	80 LQFP 112 LQFP	112 LQFP 121 MAPBGA	112 LQFP 121 MAPBGA	80 LQFP

[1] The full debug/trace interface is available only on the 112- and 121-pin packages. A reduced debug interface is bonded on the 80-pin package.

3. PŘEHLED KOMUNIKACE POMOCÍ PROTOKOLU HTTP

Zkratka HTTP znamená v překladu hypertextový přenosový protokol. Jedná se o protokol používaný ve známé komunikaci www (word wide web) hlavně při přenosu webových stránek (hypertextových dokumentů). Je ale možné jej použít pro přenos souborů daleko převyšujících velikosti běžných webových stránek.

3.1 ÚVOD DO PROTOKOLU HTTP

Původní protokol byl definován v roce 1991 a nesl označení HTTP 0.9. Podporoval pouze metodu *GET* a nepoužíval verzi protokolu. I když se dnes již nepoužívá, je možné díky zpětné kompatibilitě stránku v této verzi v prohlížeči zobrazit. Ostatně stejně jako další verze, jsou totiž zpětně kompatibilní. Dodnes používanou verzí je verze 1.0, která již hlavičku *POST* implementuje a přidává další prvky, jakými jsou metoda *HEAD* a využití MIME hlaviček. Verze HTTP 1.1, kterou podporuje tento webový server, obsahuje další vlastnosti a jsou to zejména nové metody *PUT, TRACE, CONNECT* a *DELETE*. Dokáže zpracovávat více požadavků najednou a umožňuje udržovat spojení aktivní. To znamená, že když bude formulář obsahovat nejenom text, ale také obrázky, nemusí klient pro každý obrázek navazovat nové spojení.

Základní architektura v komunikaci pomocí protokolu http je založena na komunikaci typu klient – server. V případě, že se navazuje přímé spojení protokolem TCP mezi klientem a serverem, zapíše uživatel do okna prohlížeče identifikátor objektu (URI), jejž chce prohlížet a klient nejprve z identifikátoru objektu vyřízne jméno serveru, jež přeloží za pomoci DNS na IP adresu. Poté klient naváže s tímto způsobem získanou IP adresou serveru spojení protokolem TCP.

Do takto vytvořeného kanálu vloží prohlížeč HTTP dotaz. Na který v témž spojení server odpoví HTTP odpovědí. Prohlížeč následně zobrazí odpověď uživateli. Je důležité si uvědomit, že prohlížeč zobrazuje uživateli webovou stránku, která se jako taková skládá většinou z řady objektů. Každý z těchto objektů je nutné ze serveru

stáhnout jedním HTTP dotazem. V protokolu HTTP starších verzí se pro každý dotaz vždy navazovalo nové TCP spojení. Takže prvním dotazem se stáhl základní text stránky, ve kterém byla řada dalších odkazů, které bylo nutné stáhnout pro zobrazení stránky.

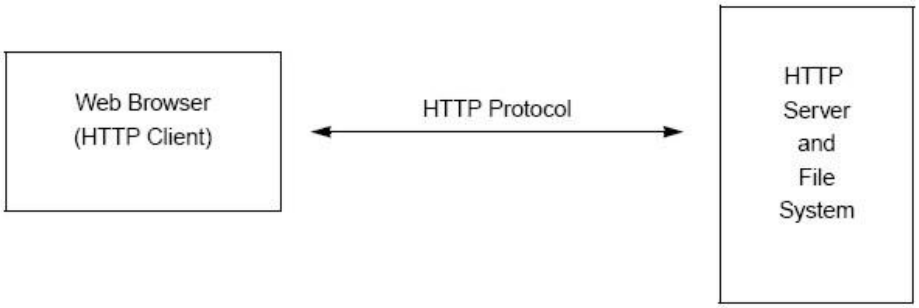
Protokol http 1.1 implicitně předpokládá, že TCP spojení bude mezi klientem a serverem navázáno jedno po celou sadu dotazů („pro celou webovou stránku“). Je možné jej uzavřít po jednom dotazu i po více dotazech. Klient může v jednom spojení odeslat více dotazů, aniž by vždy čekal na vyřízení předchozího dotazu (pipelining).

Protokol HTTP verze 1.1 implicitně předpokládá, že do vytvořeného spojení se vkládá více dotazů a odpovědí na ně. Pakliže je požadováno spojení explicitně ukončit, je třeba do záhlaví vložit hlavičku Connection: Close.

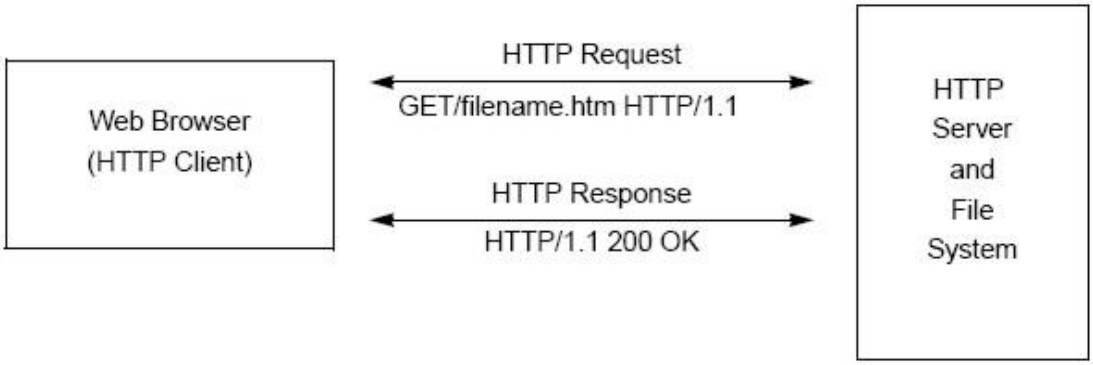
3.2 KOMUNIKACE POMOCÍ PROTOKOLU HTTP 1.1

Jak vyplývá z předchozí kapitoly, je protokol HTTP založen na principu požadavek – odpověď (obr. 3). Klient si vyžádá webovou stránku ze serveru a server mu předá odpověď v podobě webové stránky (HTML- Hyper-Text Markup Language). Protokol HTTP může být použit k posílání jakéhokoliv typu dat, včetně dat binárních. Klient si od serveru žádá soubor pomocí metody *GET* (obr. 4). Jelikož je HTTP ASCII protokol, probíhá v rámci požadavku komunikace v ASCII kódu. Server odpoví pomocí HTTP hlavičky následované obsahem požadovaného souboru. Klient může také posílat soubor pomocí metody *POST*. V rámci požadavku je komunikace HTTP v ASCII kódu. Přehled metod naleznete v tabulce č. 3 a stavové kódy tabulce č. 4.

Obr. 3. *Protokol HTTP*



Obr. 4. *HTTP požadavek / odpověď*



3.3 PŘÍKLAD KOMUNIKACE KLIENTA S WWW SERVEREM

Na následujícím příkladu si ukážeme, jak komunikace klienta se serverem probíhá.

- *GET/jmeno_souboru.htm HTTP/1.1*
  - Žádá po severu obsah souboru jmeno_souboru.htm
  - Uvádí serveru jako standard komunikace formát HTTP 1.1
- *Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjeg, application/msword*
  - Klient sděluje seznam formátů, které podporuje: x-xbitmapy, jpeg a pjeg obrázky a také podporuje dokumenty MS-WORD
- *Accept-language: en-us*
  - Podporovaný jazyk na straně klienta

- *Accept-Encoding: gzip, deflace*
  - Možné algoritmy pro dekompresi
- *User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)*
  - Prohlížeč klienta je IE6.0 běžící na platformě Windows
- *Host: www.msn.com*
- *Connection: Keep-Alive*
  - Nakonec žádá server, aby neuzavíral spojení po odeslání žádaného souboru

Server odpovídá na metodu *GET* hlavičkou následovanou různými daty nebo obsahem souboru. A následuje příklad odpovědi serveru.

- *HTTP/1.1 200 OK*
  - Protokol HTTP 1.1 je podporován a status kód 200 říká, že byl soubor nalezen (viz tabulka 3.2)
- *Server: Microsoft-IIS/6.0*
  - Klient také obdrží informace a typu a verzi serveru (Microsoft-IIS/6.0)
- *Cache-Control: no-cache*
  - Řádek říká klientovi, aby stránku neukládal (no-cache)
- *Content-Type: text/html*
  - Typ dat, která budou následovat (text/html)
- *Content-Length: 9062*
  - Udává, jaká bude délka následujících dat
- *Content-Encoding: gzip*
  - Následující data budou komprimována pomocí algoritmu gzip.

Po odeslání hlavičky HTTP serverem, následuje zaslání binárně kódovaného obsahu souboru nebo dat binárně kódovaných dat.

Tab. 2. *Porovnání funkcí knihovny MINI SOKET a knihovny BSD SOKET*

MINI SOKETY	**BSD SOKETY**
m_socket()	socket()
m_connect()	connect()
m_recv() a / nebo m_send() nebo tcp_send() a / nebo tcp_recv() - (zero copy I/O)	recv() a / nebo send()
m_close	close()
PRO SERVEROVÉ APLIKACE	
spojeno s příkazem listen	socket()
spojeno s příkazem listen	bind()
m_listen()	listen()
řízeno pomocí zpětného volání	accept()
m_recv() a / nebo m_send() nebo tcp_send() a / nebo tcp_recv() - (zero copy I/O)	recv() a / nebo send()
m_close()	close()

Tab. 3. *Metody protokolu HTTP 1.1 dle RFC2616*

METODA	**RFC2616**	**POPIS METODY**
OPTIONS	odstavec 9.2	požadavek informací
GET	odstavec 9.3	požadavek souboru nebo dat
HEAD	odstavec 9.4	totožné s GET bez těla zprávy v odpovědi
POST	odstavec 9.5	poslat data
PUT	odstavec 9.6	poslat soubor
DELETE	odstavec 9.7	smazat soubor
TRACE	odstavec 9.8	požadavek Echo
CONNECT	odstavec 9.9	rezervováno pro tunelování

Tab. 4. *Stavové kódy a jejich popis*

STATUS KÓD	POPIS
1xx	informativní odpověď, zpracování pokračuje dále
2xx	akce proběhla úspěšně
3xx	přesměrování, tj. další akce se bude týkat jiného URI
4xx	chyba klienta (např. syntaktická chyba v dotazu)
5xx	chyba serveru (např. chyba ve skriptu)

4. COLDFIRE TCP/IP STACK

Přenosový protokol TCP/IP je protokolem používaným pro internetovou komunikaci. Jeho název je složen ze dvou vrstev komunikačního stacku podle OSI modelu. Běžně používaný TCP/IP stack používá multi-taskingové zpracování, které není pro náš případ možné. ColdFire stack je integrován společně s jednoduchým operačním systémem. Tento jednoduchý cyklicky obsluhující systém může být využíván naší aplikací. Tento operační systém je nepreemptivní pracující ve dvou módech single stack neboli super smyčka (super loop) a multi stack. Mezi další vlastnosti patří interaktivní, v reálném čase modernizovatelný systém menu, uživatelské časovače a správa paměti heapu.

4.1 ÚVODEM O CODLFIRE STACKU A RTOS

TCP/IP stack potřebuje současně zpracovávat mnohonásobné procesy nebo úlohy. Tohoto dosahuje za pomoci jedné z metod: super smyčky (RTOS je zakázán) nebo multitaskingovým operačním systémem (RTOS).
V módu tzv. super smyčky je každá úloha zpracována za pomoci volání příslušné funkce. Všechny úlohy sdílejí společně jeden všeobecný stack. Výhodou tohoto módu je jeho vysoká efektivnost využití paměti a tím pádem nejmenší paměťové nároky. Použitím pouze jediného stacku nedochází k plýtvání paměťového místa. Při nevyužívání systému RTOS se také ušetří paměťové místo, které by jinak zabíraly struktury tohoto systému. Nevýhodou tohoto systému je, že je pouze statický. Úlohy nemohou být vytvářeny, ani rušeny v reálném čase. A dokonce v závislosti na architektuře může dojít k situaci, kdy mód super smyčky bude využívat větší obsazení RAM paměti, než systém RTOS. S jeho obdobou se ale můžeme setkat u méně výkonných procesorů, které nesplňují paměťové nároky operačních systémů. Spící režim není v módu super smyčky podporován. Každá funkce se na konci musí vrátit do smyčky hlavního programu.

4.2 ZÁKLADNÍ PŘEHLED VLASTNOSTÍ

Firmy InterNiche Technologies a Freescale společně vyvinuly OEM verzi Interniche optimalizovanou pro architekturu ColdFire. Produkt nazvaný ColdFire TCP/IP Lite poskytuje funkčnost NicheLite a je RFC kompatibilní. Tento stack do sebe přímo „zapouzdřuje“ následující protokoly a tím velmi usnadňuje práci s mikrokontrolérem a jeho prací v síti LAN:

- ARP - Address Resolution Protocol
- IP - Internet Protocol
- ICMP - Internet Control Message Protocol
- UDP - User Datagram Protocol
- TCP - Transmission Control Protocol
- DHCP klient - Dynamic Host Configuration Protocol
- BOOTP - Bootstrap Protocol
- TFTP - Trivial File Transfer Protocol

Interniche stack obsahuje dva operační systémy:
SuperLoop: velmi rychlý, stále dokola zpracovávající různé úlohy a aplikace v modelu „run-to-completion“. Odtud je každá úloha spouštěna souvisle ve hlavní smyčce programu a není tady tedy možnost žádného přepínání úloh při vysokém zatížení a dochází k problémům s nízkou účinností během špatně odladěných úloh stacku.
NicheTask: spolupracující multi taskingový plánovač. V této architektuře každá úloha má svůj vlastní call-stack a dobrovolně odevzdává kontrolu zpět hlavnímu plánovači.
Plné verze programů NicheStack IPv4 a IPv6 obsahují velkou řadu protokolů pro Internet, Intranet a připojení LAN při využití sítí IPv4 a IPv6. Tyto přenositelné TCP/IP stacky jsou kompletem SDK pro vývojáře síťových zařízení. IP vrstva může být konfigurována jako standardní klient, IP router nebo jako server pro domácnost. Firma InterNiche, pro usnadnění práce, také nabízí přídavné volitelné moduly s názvem InterNiche Protokolové moduly pro ColdFire TCP/IP Lite. Volitelné

protokolové moduly InterNiche byly vyvíjeny pro potřeby nízko rozpočtových embedded systémů. Výsledkem tohoto úsilí je modulární, funkčně komplexní systém vyžadující pouze nejnutnější systémové a paměťové požadavky. Tento systém je vyladěn jak pro výkon, tak i pro přenositelnost.
V současné době nabízené moduly jsou:

- Email
- FTP
- HTTP Server
- PPP
- PPPoE
- SNMP
- Telnet
- Security moduly
- Routing / gateway
- DHCP Server
- DNS server

4.3 REFERENČNÍ OSI MODEL

Protokoly TCP, IP a HTTP jsou jedněmi z mnoha protokolů, které jsou vytvořeny podle struktury referenčního modelu OSI. Účelem tohoto modelu bylo definovat strukturu, která stanovuje logické úkoly komunikace požadované pro přemisťování informací mezi počítačovými systémy. Základním předpokladem modelu OSI je definovat a seskupit logické funkce toku informací mezi systémy, aniž by se pokoušel popisovat detaily každé z funkcí.
Horní vrstvy modelu OSI (5 – 7) souvisejí s aplikacemi a obvykle jsou implementovány v softwaru. Dolní vrstvy (1 – 4) souvisejí s přenosem informací v síti a mohou být implementovány v hardwaru, softwaru anebo firmwaru.

Přehled vrstev referenčního OSI modelu:

- Vrstva číslo 7 – Aplikační (HTTP, SMTP, POP3, TFTP)
- Vrstva číslo 6 – Prezentační (Berkeley Socket Interface, XTI)
- Vrstva číslo 5 – Relační (Berkeley Socket Interface, XTI)
- Vrstva číslo 4 – Transportní (TCP, UDP)
- Vrstva číslo 3 – Síťová (IP, ARP, ICMP)
- Vrstva číslo 2 – Datová - MAC (Ethernet, PPP)
- Vrstva číslo 1 – Fyzická - PHY (RS232, 10Base-T, DSL)

4.4 PROTOKOL TCP

V předchozí části popisovaný protokol HTTP popisuje výměnu dat probíhající mezi serverem a klientem. Pro tuto komunikaci využívá takzvaný přenosový protokol. HTTP může být transportováno jakýmkoliv protokolem podporujícím obousměrný přenos ASCII kódu. Pro přenos po internetu se používá protokol TCP/IP (Transport Control Protocol / Internet Protokol).

TCP/IP je protokol založený na vzájemném spojení. Protokol HTTP nemá žádné mechanizmy pro potvrzení doručení (ACK) nebo návratu a spoléhá se spíše s touto službou na transportní protokol. TCP část protokolu z TCP / IP uskutečňuje navázání spojení, obstarává zasílání potvrzovacích zpráv a zabezpečuje návrat při chybné komunikaci. Protokol HTTP nepoužívá žádné funkce zaručující přenos dat ze serveru k hostu. Toto provádí výše zmíněný protokol TCP.

Protokol IP (Internet Protokol) poskytuje mechanizmus mnohanásobného TCP spojení vztahující se k portům. Pro každé IP spojení je dostupných 65 534 portů. Každé IP spojení má unikátní IP adresu a stejně tak každý uzel v internetu má svoji IP adresu. IP adresa se sestává ze čtyř bajtů zapsaných pomocí tečkového zápisu.

IP porty jsou přiděleny na základě protokolu, který je využívá. Protokol HTTP standardně používá port 80. To znamená, že standardní HTTP server naslouchá spojení na portu 80. Samozřejmě může být HTTP serveru přiřazen jiný, nestandardní port, ale webové prohlížeče používají tento port jako výchozí pro tuto komunikaci.

Na Interniche TCP / IP stack je odkazováno jako na stack (hromadu), protože samotný protokol TCP / IP je vyvinut na jiném protokolu, založeném na dalším (dochází tedy k hromadění).

4.5 PREZENTAČNÍ VRSTVA

Na vrcholu TCP / IP stacku nalezneme podle OSI modelu takzvanou prezentační vrstvu. Tato vrstva slouží vývojářům jako propojení mezi psaným programem a TCP / IP stackem. Jediné co je pro vývoj takové aplikace třeba znát je, jak používat prezentační vrstvu pro použití stacku. V operačních systémech jako je Windows nebo Linux je v této vrstvě obvykle používáno BSD soketové rozhraní.
BSD soketové rozhraní je rozsáhlý aplikační interfejs (API). Server HTTP používá pouze několik z dostupných příkazů. Soket je základním stavebním prvkem spojení a může být dopraven skrze komunikační kanál z jednoho konce na druhý. Veškerá komunikace probíhá pomocí soketů. Server otvírá a naslouchá sokety a klient otvírá a připojuje sokety. Použitím TCP vrstvy stačí dva sokety na otevření a navázání obousměrného spojení na lince. Po navázání spojení již jednoduše komunikují server a klient za pomocí soketů.
HTTP server používá soketové spojení na vytvoření „naslouchacího soketu“ na portu 80. IP vrstva má přiřazenu jedinečnou adresu, kterou získá během inicializace TCP / IP stacku. Při otvírání a naslouchání soketu na portu 80, otvírá a naslouchá HTTP server na jedinečné adrese IP stacku.
Klient (webový prohlížeč) otevírá spojovací soket, specifikuje IP adresu serveru a HTTP port (80). Poté, co server akceptuje spojení, dojde výměnou dvou soketů k navázání komunikace v protokolu TCP. Po přijmutí této komunikace komunikují server a klient pomocí funkcí *recv()* a *send()*. Ačkoliv probíhá komunikace TCP / IP v paketech, aplikace nepotřebuje rozdělovat data na pakety, tuto službu přebírá TCP / IP stack. Aplikace pouze jednoduše zasílá části dat do fronty stacku. Stack rozdělí sám data na pakety a provede veškeré operace s vyrovnávací pamětí.
ColdFire TCP / IP stack nepodporuje všechny Berkeleyovo sokety. Používá takzvaný mini soketový interfejs (tab. č. 2).

Největší rozdíl je na straně serveru. S mini sokety stačí uživateli pouze jedna funkce *m_listen()* k otevření a naslouchání na soketu. Z tabulky č. 2 je patrné, že mini sokety nepodporují funkci *accept()*. Místo toho, když se klient připojí na server, stack použije funkci zpětného volání, aby serveru HTTP oznámil, že je klient připojen.

4.6 PŘÍKLAD SUPER SMYČKY A RTOS

Na na obrázku č. 5 je názorně vidět rozdíl mezi super smyčkou a stackem. Na následujícím příkladu je tento rozdíl ukázán na krátkém programovém kódu.

```
void emg_superloop(void)
{
While(1)
        {
        kbdio();        // volání funkce menu
        packet_check(); // volání síťového stavového automatu
        inet_timer();   // kontrola vypršení času časovačů
        task1();        // volání uživatelské úlohy 1
        task2();        // volání uživatelské úlohy 2
        task3();        // volání uživatelské úlohy 3
        task4():        // volání uživatelské úlohy 4
        }
}

struct uloha
{
        struct uloha *tk_next; /* ukazatel na další úlohu */
        stack_t *tk_fp;        /* ukazatel na probíhající úlohu */
        char *tk_name;         /* název úlohy */
        int tk_flags;          /* příznak je nastaven, když je úloha naplánována */
        unsigned long tk_count;      /* číslo wakeup */
        stack_t *tk_guard;           /* ukazatel dna stacku */
```

```
    unsigned tk_size;          /* velikost stacku */
    stack_t *tk_stack;         /* ukazatel vrcholu stacku */
    void *tk_event;            /* událost budící zablokovanou úlohu */
    unsigned long tk_waketick; /* čas k probuzení úlohy */
};
```

Systém RTOS vykonává veškeré operace v dynamickém režimu. Úlohy mohou být vytvářeny a rušeny za běhu aplikace. Pro každou novou úlohu je vytvořen nový stack alokací paměťového místa z heapu. Při ukončení této úlohy je toto místo zpět heapu navráceno. Když budeme na využití místa RAM paměti pohlížet z tohoto úhlu, bude se nám jistě zdát efektivnější.

Velikost alokovaného stacku pro úlohu je konstantní a je zadána již při kompilaci programu. Stack musí být dostatečně veliký nejenom pro potřeby samotné úlohy, ale také pro přerušení, která systém používá. Opomenutí tohoto faktu by velice rychle vedlo ke zkolabování celého programu.

Při použití systému RTOS má každá úloha svoji blokovou kontrolu. TCB jsou spojovány podle seznamu spojení. Struktura TCB je deklarována v souboru *task.h*. Tento jednoduchý RTOS systém nepodporuje rozlišování priorit u úloh. Program jednoduše přidá další TCB do seznamu a spouští úlohu ukazující na TCB, pokud je úloha připravena na spuštění (není ve spícím režimu). Z toho vyplývá, že RTOS není preemptivní, k přepínání úloh dochází, pouze když probíhající úloha předá řízení. K tomu slouží funkce *tl_block()* nebo uspání úlohy *tk_sleep()*.

4.1 KONFIGURACE

Konfigurace módu RTOS se provádí v souboru *ipport.h*. Nacházejí se zde obě verze definicí RTOS, které ale nemohou být použity současně. Když je proměnná *superloop* nastavena na 1, pracuje RTOS v režimu super smyčky. Když je nastavena na 1 proměnná *INICHE_TASKS*, je aktivován více stackový režim, jako je tomu na následujícím příkladu.

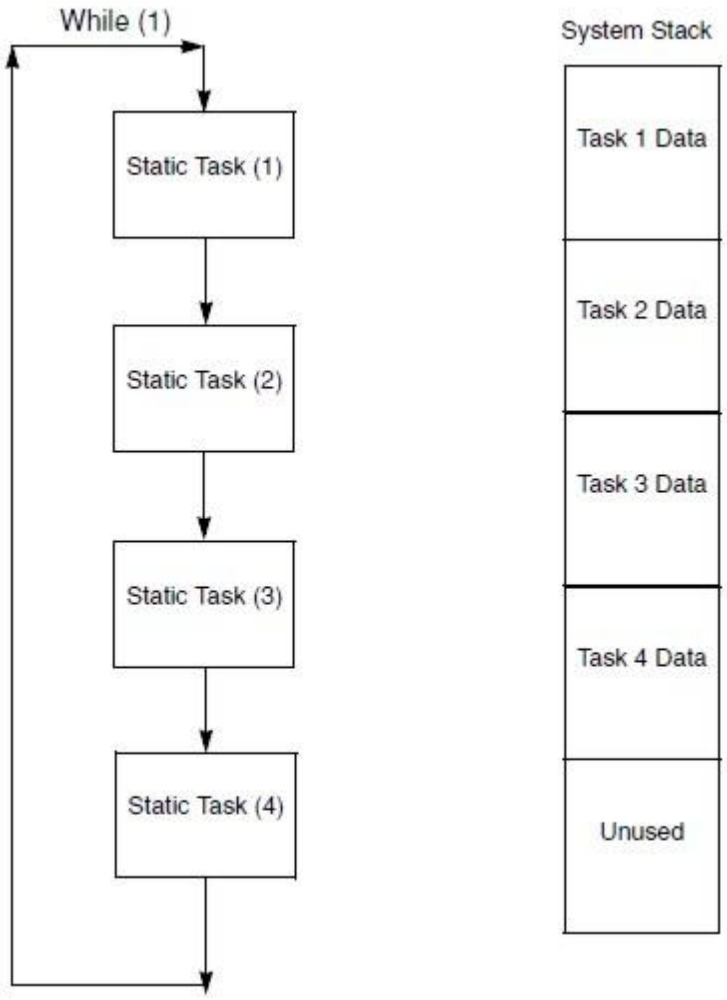
```
// #define  SUPERLOOP  1
#define  INICHE_TASKS  1          /* InterNiche multitasking system */
```

Při aktivaci RTOS je potřeba nastavit ještě další parametry v souboru *osport.h* jako je velikost stacku pro každou úlohu. Každý stack může být inicializován na různou velikost. Díky tomuto může pracovat kterákoliv úloha i během přerušení. Přerušení je uloženo v stacku právě probíhající úlohy. Tato technika ovšem vyžaduje mnohem větší nároky na RAM paměť a je nutné mít dostatek místa u všech stacků alokovaných pro úlohy. Každý stack musí být větší, než samotná úloha potřebuje. Důsledkem toho je plýtvání pamětí RAM.
Dále je potřeba mít definovánu globální proměnnou (v *main.c*), která bude periodicky inkrementována pro odpočítávání spícího režimu. Tuto funkci zajišťuje časovač ColdFire. Přerušení časovače (*timer_isr*) je vyvoláno a znovu inicializováno funkcí *clock_init()* v *main.c*.

4.2 RTOS API FUNKCE

Žádná funkce API není přístupná při použití super smyčky. Po vykonání úlohy se vrací program zpět a může zpracovat další úlohu. S aktivovaným systémem RTOS máme více možností. Každá úloha má dva možné stavy: spaní nebo vykonání úlohy. Úloha tedy může spát nebo čekat na událost. Úloha označená jako „spící do události“ může znovu probíhat pouze, když ona událost nastane.
Úloha ve spícím stavu je systémem spouštění úloh přeskočena a nemůže být znovu spuštěna, dokud na ni nedojde znovu řada podle plánovače úloh. Pokud je úloha spustitelná, stane se to až tehdy, kdy na ni přijde řada podle plánu cyklického spuštění úloh. Stav následující úlohy je testován funkcí *tk_block()*.
Systém RTOS nepotřebuje ke své funkci přerušení. Stačí, když časovač pravidelně inkrementuje již zmíněnou proměnnou, na kterou ukazuje programový spouštěč ve funkci *tk_block()*, určující, jak dlouho úloha spala. Když jde úloha spát, hodnota proměnné udávající čas spánku je uložena v TCB. Když je zavolána funkce *tk_block()*, kontroluje u další úlohy TCB, jestli již nebylo dosaženo hodnoty uložené v TCB.

Obr. 5. *Struktura super smyčky*



4.3 PŘEHLED API FUNKCÍ

Rozhraní API pro RTOS obsahuje celkem devět funkcí.

- *task * tk_init(stack_t * base, int st_size)*
  Inicializuje RTOS a vytvoří první úlohu. Adresa dna stacku a jeho velikost jsou předány jako parametry.
  Jméno prvního stacku je „Main“.
- *task * tk_new (task * prev_tk* // předchůdce nové úlohy
  *int (*start)(int)* // ukazuje, kde začala nová úloha
  int stksiz // v bytech uvedená velost stacku nové úlohy

name // jméno úlohy jako string

int arg) // argumenty úlohy

Funkce *tk_new()* vytvoří novou úlohu. Nový TCB je vložen za ukazatel na TCB pomocí funkce *prev_tk*. Uživatel může vložit novou úlohu kdekoliv v TCB cyklu při použití tohoto parametru. Zakrývací makro *TK_NEWTASK* (definované v *osport.h*) zavolá *tk_new()* s nastaveným parametrem *prev_tk* na TCB právě probíhající úlohy.

Počáteční parametr je ukazatel na funkci nové úlohy. Pomocí parametru *stksiz* definujeme velikost stacku úloh. Stack pro novou úlohu je přidělen z heapu. Parametry jsou pojmenovány podle nové úlohy. Hodnoty parametrů jsou předány nové funkci pomocí celočíselných argumentů při startu.

- *void tk_block(void)*

  funkce *tk_block()* nejdříve ověří, zda není porušen stack právě probíhající úlohy pomocí ochranného čísla. Pak prochází připojený TCB seznam a hledá, která z dalších úloh je připravena na spuštění. Poté si vynutí přepnutí úlohy voláním assembler funkce *tk_switch* definované v *tk_util.s*. Přerušení jsou během přepínání úlohy zakázána.

- *void tk_exit(void)*

  Funkce *tk_exit()* nastaví právě probíhající funkci na vymazání při příštím přepnutí. Poté si vynutí přepnutí úlohy. Úloha je vymazána během přepínání.

- *void tk_kill(uloha * uloha_na_ukončení)*

  tk_kill() nastaví ukazatel na úlohu pomocí parametru *uloha_na_ukončení* na ukončení. Jestliže je označena právě probíhající úloha, je ukončena po dokončení bloku. V případě, že je označena jiná než právě probíhající úloha, je ukončena okamžitě.

- *void tk_wake(uloha * tk)*

  tk_wake() funkce přivádí úlohu, na kterou ukazuje ukazatel *tk*, do běžícího stavu. Úloha se ovšem nespouští okamžitě, spíše se označí stavem spuštění a je spuštěna, jakmile na ni přijde řada v cyklu.

- *void tk_sleep (long délka_čekání)*
  uspí probíhající úlohu, na dobu udanou počtem impulzů v proměnné délka_čekání
  probíhající_TCB->tk_waketick = cticks + délka_čekání;
  probíhající_TCB->tk_flags &= ~TF_AWAKE; nuluje příznak probuzení
  tk_sleep() voláním *tk_block()* způsobí okamžité přepnutí úlohy
- *void tk_ev_block(void * událost)*
  tk_ev_block() přepne probíhající úlohu do nečinného stavu (spaní), dokud nenastane zadaná událost. Událost je 32 bitová hodnota.
- *void tk_ev_wake(void * událost)*
  prochází TCB, porovnává parametr události s parametrem událostí zadaným v TCB struktuře. Pokud se parametry shodují, je úloha označena pro spuštění. Události poskytují komunikační mechanizmus mezi úlohami. Stack používá události pro pomoc s blokováním soketů. Zatímco soket čeká na data je blokován, dokud data nedorazí. Stack uspí volající úlohu, dokud nenastane definovaná událost (adresa soketu). Když na tomto soketu dorazí všechna data, stack probudí zablokovanou úlohu za pomoci zaslání události.

4.4 KONFIGURACE TCP/IP STACKU

TCP/IP stack se konfiguruje v hlavičkovém souboru *ipport.h*. Použitím jednotlivých doplňků definovaných pomocí maker narůstá i potřeba paměťového místa. Je důležité nepovolovat moduly, které nejsou pro danou aplikaci nutné, jinak dojde k tomu, že budou zbytečně plýtvány prostředky pro zpracování nepotřebných částí kódu.

Jak již bylo dříve uvedeno, statické aplikace jsou méně náročné na velikost paměti RAM, jestliže není pro ně aktivován *INICH_TASKS* a používají pouze super smyčku. Pro tento případ je potřeba neaktivovat *INICHE_TASKS* a nastavit super smyčku (superloop) na 1.

Mezi další soubory potřebné pro konfiguraci patří *m_ipnet.c*, ve kterém definujeme IP adresu stacku v případě, že nepoužíváme DHCP klienta (proměnná

DHCP_CLIENT = 0). IP adresa je uložena v proměnné *netstatic[]* a musí být vynulována při použití DHCP klienta.

V případě použití DHCP není dokončena inicializace TCP/IP stacku dokud není dokončena DHCP transakce. Ta probíhá tak, že funkce *netmain_init()* v modulu *allports.c* zavolá funkci *dhc_setup()* v *dhcsetup.c* a ta spustí DHCP protokol, který se spojí s DHCP serverem a získá od něj potřebnou adresu a ostatní potřebné síťové parametry. Pro náš případ nebude výhodné DHCP klienta používat a tak bude potřeba nastavit IP adresu předem. Taktéž je nutné nastavit MAC adresu pomocí *netstatic[]* a *mac_addr_fec[]*. Pole *netstatic[]* je deklarováno v souboru *m_ipnet.c* a pole *mac_addr_fec[]* je deklarováno v souboru *ifec.c*.

Adresu IP 192.168.1.99 nastavíme následovně.

```
netstatic[0].n_ipaddr = (0xC0A80163);
netstatic[0].n_defgw = (0xC0A80101);
netstatic[0].snmask = (0xffffff00);
```

A takto nastavíme adresu MAC na hodnotu *0x00badbad0102*:

```
tmp = 0x00badbad;
mac_addr_fec[0] = (u_char)(tmp >> 24);
mac_addr_fec[1] = (u_char)(tmp >> 16);
mac_addr_fec[2] = (u_char)(tmp >> 8);
mac_addr_fec[3] = (u_char)(tmp & 0xff);

tmp = 0x01020304;
mac_addr_fec[4] = (u_char)(tmp >> 24);
mac_addr_fec[5] = (u_char)(tmp >> 16);
```

Tyto struktury musí být nakonfigurovány předtím, než je inicializován stack voláním funkce *netmain_init()* v modulu *allports.c*.

Mezi další možnosti, které v našem ale nebudou použity, je DNS klient. DNS klient komunikuje s DNS serverem za účelem překladu jména domény na její IP adresu.

4.5 API FUNKCE TCP/UDP/IP STACKU

Mini sokety API jsou vyvinuty tak, že se velmi podobají BSD API soketům, přičemž potřebují mnohem méně systémových prostředků, než BSD API sokety. Základní rozdíl je v tom, že pasivní spojení jsou obsloužena během jediného volání funkce *m_listen()* oproti sekvenci příkazů používaných v BSD – *bind() - listen() - accept()*. Jednotlivé příkazy jsou:

- *char *ip_startup()*
  Inicializuje TCP/IP stack. Vrací nulu, pokud proběhla inicializace v pořádku, jinak vrací v řetězci chybu.
- *int input_ippkt(PACKET paket, int délka)*
  Zavolána jakmile ISR obdrží paket z hardwaru. Tato funkce vkládá přijatý ethernetový paket do stacku. Vrací vždy nulu. Typ paket je ukazatel ne strukturu netbuf.

```
struct netbuf
{
        struct netbuf * next;   // odkaz na frontu
        char * nb_buff;         // začátek hrubého bufferu
        unsigned nb_blen;       // délka hrubého bufferu
        char * nb_prot;         // začátek protokolu / dat
        unsigned nb_plen;       // délka protokolu / dat
        long nb_tstamp;         // časová známka paketu
        struct net * net;       // rozhraní, kterým paket dorazil
        ip_addr fhost;                  // IP adresa paketu
        unsigned short type;    // IP==0800 vyplněno v recever(rx) nebo v
netlayer.(tx)
        unsigned inuse;         // používá odpočet pro klonování bufferu
        unsigned flags;         // bitobá maska PKF_ defines
        char * m_data;          // ukazatel na TCP data v nb_buff
        unsigned m_len;         // délak m_data
        struct netbuf * m_next; // sockbuf que odkaz
        struct ip_socopts *soxopts; // socket options */
```

};

- *void packet_check( )*
  Stavový automat stacku TCP/IP. Musí být periodicky volán úlohou nebo ze super smyčky.

- *void dhc_setup()*
  Zavolaná po funkci *ip_startup()* k nastartování DHCP klienta. Stack nemůže být použit, dokud není dokončena DHCP transakce.
- *M_SOCK m_socket()*
  Alokuje strukturu soketu. Předdefinovaný soketu je zablokován. Vrací strukturu *MSOCK* v případě úspěchu, jinak *NULL*.
- *int m_connect(M_SOCK so, struct sockaddr_in * sin, M_CALLBACK(name))*
  Začne připojovací proces k serveru. Pokusí se o spojení s IP adresou a portem specifikovaných ve struktuře *sockaddr_in*. Jestliže je soket označen jako zablokovaný, *m_connect()* funkce se nevrátí, dokud nevyprší čas definovaný v *TCPTV_KEEP_INIT (v mtcp.h)*. Výchozí hodnota je 75 sekund. Jestliže je soket označen jako neblokující (pomocí funkce *m_ioctl()*), potom vrací funkce *m_connect() EINPROGRESS*. A také u soketu označeného jako neblokující, parametr signálu *M_CALLBACK* ukončí spojení zavoláním funkce *M_CALLBACK*.
  Funkce *m_connect()* vrací kódy chyb uvedené v souboru *msock.h*. Datový typ *MSOCK* je ukazatel na *msocket* strukturu.

```
struct msocket
{
        struct msocket * next;          // spojení s frontou
        unshort lport;                  // IP/port číslo portu spojení, místní port
        unshort fport;                  // vzdálený (cílový) port
        ip_addr lhost;                  // místní IP adresa
        ip_addr fhost;                  // cílová IP adresa
        struct tcpcb * tp;              // zpětný ukazatel na tcpcb
```

```
struct m_sockbuf sendq;      // řada paktů k odeslání, včetně nepotvrzených
struct m_sockbuf rcvdq;      // pakety přijaté, ale ne doručené do app
struct m_sockbuf oosq;       // pakety přijaté mimo pořadí
int error;                   // poslední chyba z BSD seznamu
int state;                   // bitová maska SS_ hodnoty ze sockvar.h
int so_options;              // bitová maska SO_ vlastností ze socket.h
int linger;                  // čas vytrvání, když je nastaven SO_LINGER
M_CALLBACK(callback);        // soket, zpětné volání rutiny
NET ifp;                     // interface pro pakety při spojení
char t_template[40];  // tcp šablona záhlaví, ukazatel, pomocí tp->template
void * app_data;             // užíváno soketovou aplikací
};

struct sockaddr_in
{
        short sin_family;
        u_short sin_port;
        struct in_addr sin_addr;
        char sin_zero[8];
};
```

Struktura *sockaddr_in* je velmi intenzivně používána v API TCP/IP stacku.

- *M_SOCK m_listen (struct sockaddr_in * sin, M_CALLBACK(name), int * error)*

  Funkce *m_listen()* vytvoří svůj vlastní soket zavoláním funkce *m_socket()* a naslouchá spojení vzdáleného klienta. Použitím struktury *sockaddr_in*, může být naslouchací soket přidělen přímo portu, na kterém naslouchá, podobně naslouchá i na IP adrese. Když je IP adresa nulová, potom *m_listen()* bude naslouchat jakékoliv adrese. Pokus server akceptuje pouze spojení z jedné zadané IP adresy je tato adresa nastavena v *sin_addr*. Pokud server přijímá

spojení z jakékoliv adresy (standardní způsob chování), je *sin_addr* nastavena na nulu. Funkce zpětného volání je zavolána při dosažení spojení.

socket->fhost = sockaddr_in->sin_addr; // vzdálená adresa IP

socket->lport = sockaddr_in->sin_port; // místní port

- *int m_close(M_SOCK so)*
  Uzavře jakékoliv spojení na soketu a uvolní jej.
- *int m_send(M_SOCK so, char * data, unsigned datalen)*
  Funkce je povolena, když je definováno makro *BSDISH_SEND*. Pracuje podobně jako funkce soketu BSD *send()*.
  Proměnná data je ukazatel na pole k odeslání a *datalen* je délka tohoto pole.
  Vrací počet úspěšně odeslaných bytů, nebo -1 v případě chyby.
- *int m_recv(M_SOCK so, char * buf, unsigned buflen)*
  Funkce je povolena, když je definováno makro *BSDISH_RECV*. Pracuje podobně jako funkce soketu BSD *recv()*.
  Parametr *buflen* udává maximální počet bytů, které může stack uchovat v *buf* poli. Pole *buf* je zaplněno daty ze stacku. *Buf* nemusí nutně dosahovat hranice paketu. Buf pole může obsahovat jenom část z paketu nebo násobků paketu. Funkce *m_rev()* a *m_send()* jsou takzvané průtokové funkce, kde data nemají hranice.
- *int m_ioctl(M_SOCK so, int option, void * data)*
  Používá se pro různá nastavení soketů.
  Parametr *option* je pro výběr vlastností a parametr data pro jejich hodnotu.
  Vlastnosti definované v *msock.h*:
  SO_NONBLOCK – nastaví soket jako neblokující
  SO_BIO – nastaví soket jako blokující
  SO_NBIO – nastaví / vynuluje blokování
  SO_DEBUG – nastaví soket do ladícího módu, při použití makra
  DO_TCPTRACE – TCP je umožněno použití ladících zpráv
  SO_LINGER – nastaví čekací dobu před uzavřením soketu, pokud je nastavena 0, bude soket uzavřen okamžitě se zavoláním funkce *m_close()*.

4.6 NÁVRATOVÉ KÓDY TCP/UDP/IP STACKU

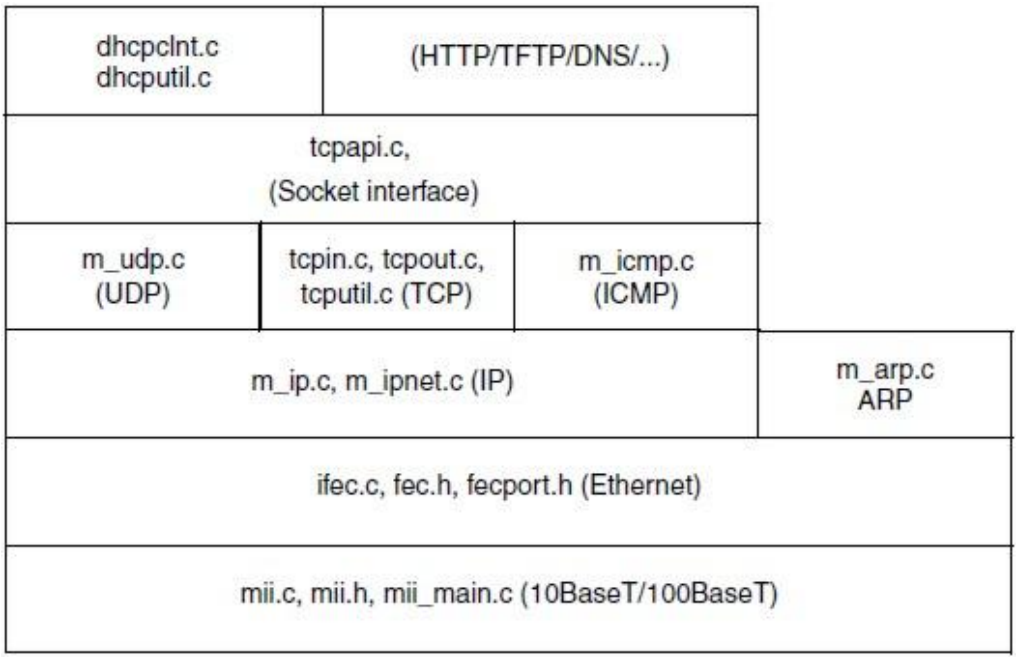
- *ENOBUFS* - není dostupný paketový bufer požadované velikosti
- *ETIMEDOUTTCP* - vypršel čas, nepřišlo ACK, není spojení
- *EISCONN* - *m_connect()* nastane chyba, je-li již soket připojen
- *EOPNOTSUPP* - *m_ioctl()* nastane chyba, není-li vlastnost podporována
- *ECONNABORTED* - nepoužito
- *EWOULDBLOCK* - indikuje, že nezablokovaný soket bude zablokován. Tato chyba nastává, když je soket nastaven na nezablokovaný a bude zavolána funkce *m_recv(),* když stack neobsahuje již žádná data.
- *ECONNREFUSED* - Spojení bylo serverem odmítnuto
- *ECONNRESET* - spojení resetováno klientem
- *ENOTCONN* - byl udělán pokus os komunikaci se soketem, který není připojen.
- *EALREADY* - nepoužito
- *EINVAL* - soket vstupující do API není platný a bude uzavřen
- *EMSGSIZE* - nepoužito
- *EPIPE* - chyba z *tcp_send()*, hostitel ukončil spojení
- *EDESTADDRREQ* - nepoužito
- *ESHUTDOWN* - Soket odpojen
- *ENOPROTOOPT* - nepoužito
- *EHAVEOO* - nepoužito
- *ENOMEM* - chyba při alokaci paměti pro soket nebo jiné požadované struktury
- *EADDRNOTAVAIL* - Muti-castová adresa nebyla nalezena
- *EADDRINUSE* - Muti-castová adresa se používá
- *EAFNOSUPPORT* - nepoužito
- *EINPROGRESS* - neblokující chyba indikující stejné spojení (*m_connect()*)
- *ELOWER* – nepoužito

4.7 TOK PAKETŮ TCP/IP STACKEM

Základní schéma toku paketů je na obrázku č. 6. Když je paket přijat, předá ho FEC ISR do stacku voláním funkce *input_ippkt()* v souboru *ifec.c* s ukazatelem na paket a délkou paketu. Paket je vložen do vstupní řady paketu voláním funkce *putq()* v modulu *q.c*. Poté je zavolána funkce *SignalPktDemux()*, která probudí TCP/IP stack.

Funkce *packet_check()* v souboru *allports.c* musí být zavolána při pravidelném běhu buď úlohy, nebo super smyčky (obr. 7). Tato funkce dále zavolá *pktdemux()* v souboru *m_ipnet.c*, aby došlo ke skutečnému zpracování paketu.

Obr. 6. *Tok paketů TCP/IP stackem*

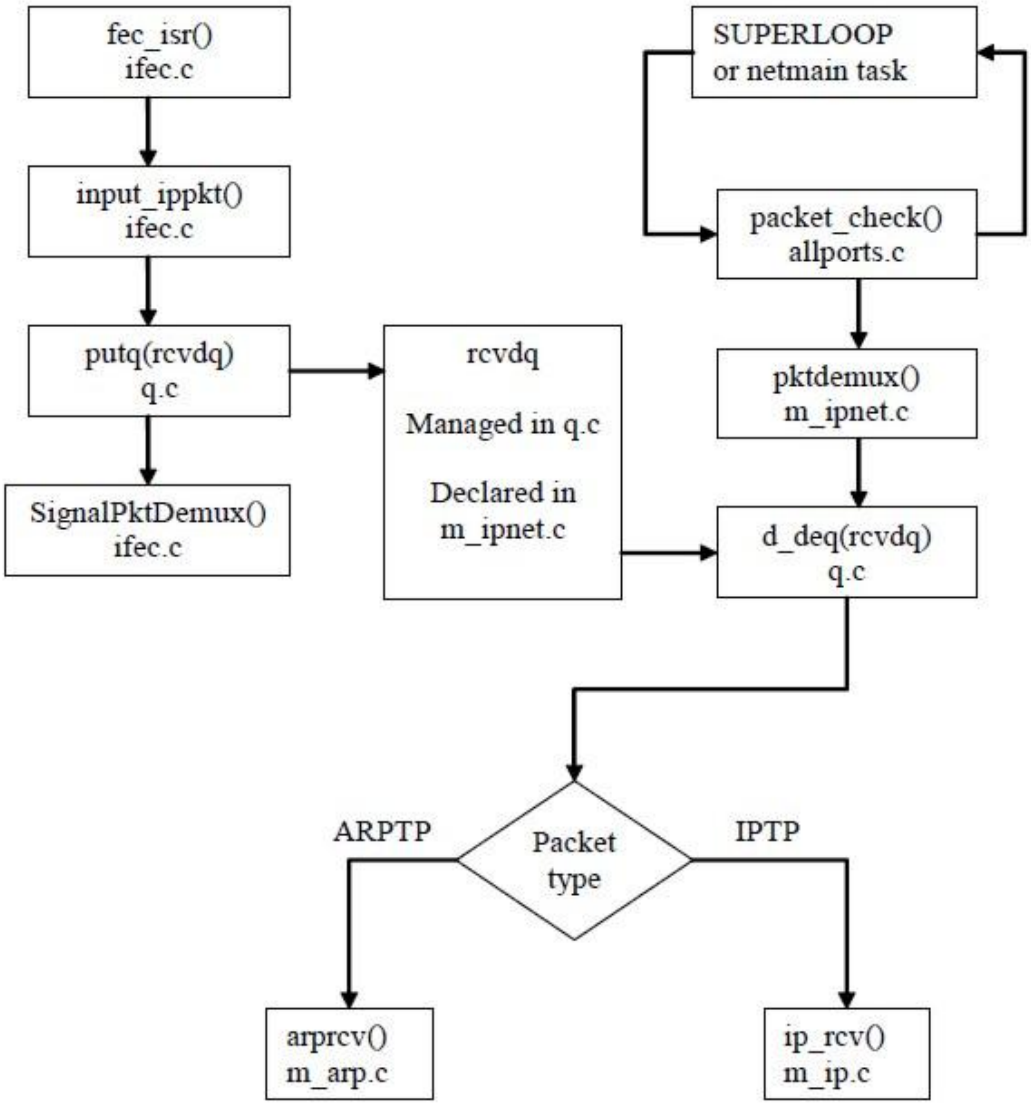


Funkce *pktdemux()* prohlédne ethernetové záhlaví, aby zjistila typ paketu. Pakety ARP jsou poslány funkcí *arprcv()* na zpracování, zatímco IP pakety jsou poslány *ip_rcv()* funkci, která požadované informace tomuto zařízení zašle. ARP odpovědi z jiného zařízení jsou reakcí na ARP požadavky. Tyto odpovědi jsou uchovávány v ARP tabulce.

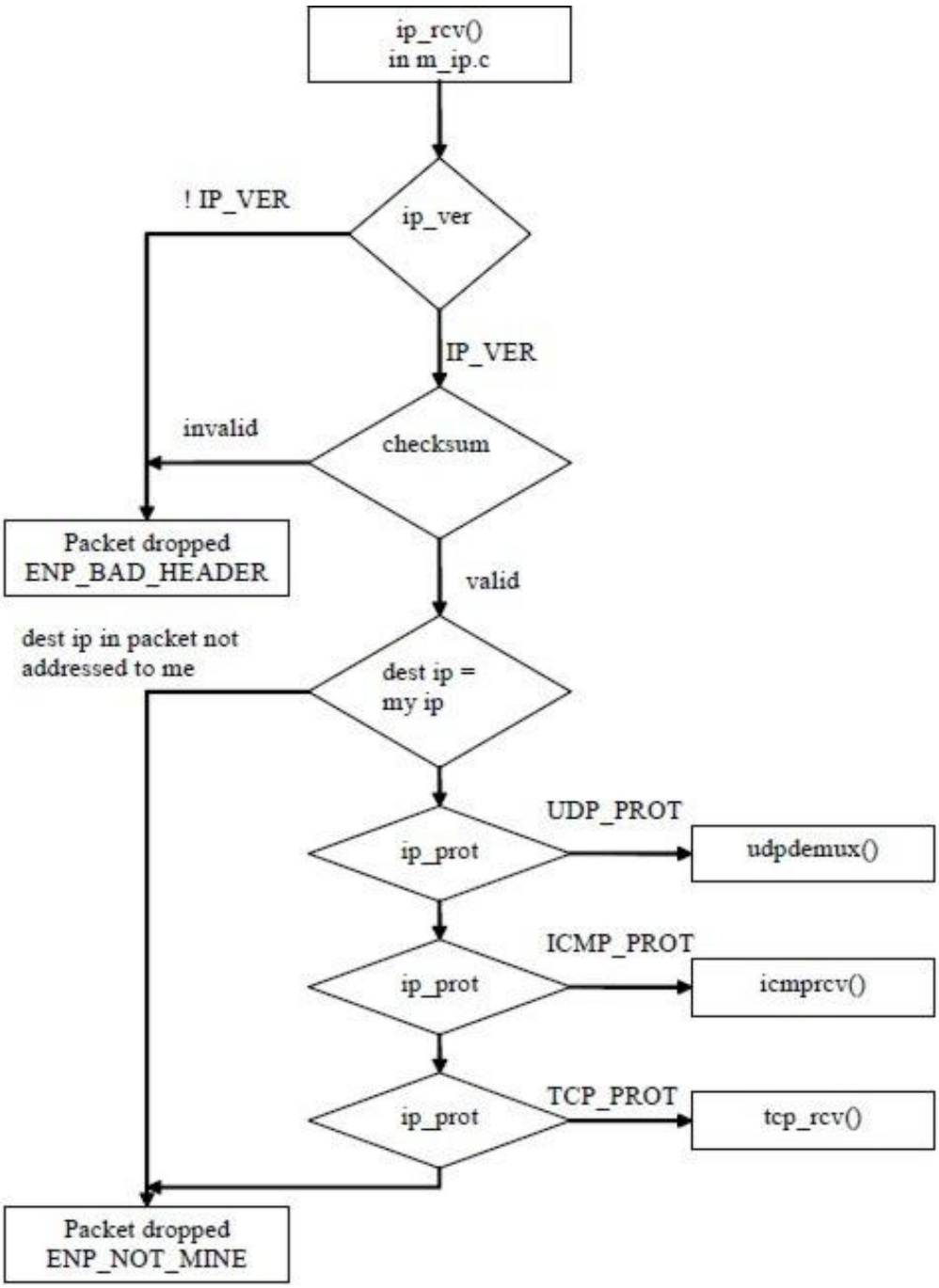
Jestliže je paket typu IPTP, potom je zavolána funkce *ip_rcv()* v souboru *m_ip.c*. Tato funkce odfiltruje IP vrstvu paketu. Nejprve kontroluje, jedná-li se o paket správného typu. Volná verze stacku podporuje pouze IPV4, pakety protokolu IPV6 jsou zahazovány. Potom testuje kontrolní součet IP vrstvy paketu. Pokud je chybný, je paket zahozen. Nakonec je zkontrolována adresa paketu, zda je paket adresován nám. Ty, které nejsou, jsou zahozeny. Zde by tedy byla možnost pro případné filtrování komunikace na základě IP adresy. Tedy po kontrole typu, kontrolního součtu a IP adresy prochází paket zásobníkem v závislosti na poli protokol v záhlaví IP.

ColdFire TCP/IP stack podporuje tři protokoly: UDP,ICMP a TCP. Uživatelské Datagramové Pakety (UDP) jsou zpracovány funkcí udpdemux(), ICMP pakety potom funkcí *icmprcv()* a TCP pakety funkcí *tcp_rcv()*. Schéma těchto pochodů naleznete na obrázku č.8.

Obr. 7. *Blokové schéma zpracování paketu*



Obr. 8. *Kontrola paketů*



5. JEDNODUCHÝ PŘÍKLAD IMPLEMENTACE API FUNKCÍ TCP/IP STACKU

Na následujícím příkladu je ukázána „holá“ implementace API funkcí stacku, začínající vytvořením soketu, na krerém aplikace naslouchá komunikaci. Akceptováním soketu při jeho úspěšném otevření a následném příjmu nebo odeslání TCP dat. Po dokončení komunikace je soket uzavřen.

Vytvoření naslouchacího soketu:

```
emg_http_sin.sin_addr.s_addr = (INADDR_ANY);
emg_http_sin.sin_port = (PORT_NUMBER);
emg_http_server_socket = m_listen(&emg_http_sin, freescale_http_cmdcb, &e);
```

Příjem spojení:

```
switch(code)
{
        // soket je kompletně otevřen
        case M_OPENOK:
        msring_add(&http_msring, so);
        break;
};
```

Příjem TCP dat.

```
length = m_recv( freescale_http_sessions[session].socket, (char *)buffer, RECV_BUFFER_SIZE );
```

Odeslání TCP dat

```
bytes_sent = m_send( freescale_http_sessions[session].socket, data, length);
```

Uzavření soketu

```
j = m_close(so);
```

6. VLASTNÍ POPIS WEBOVÉHO SERVERU

Ačkoliv jak je výše uvedeno, je jednodušší vytvořit aplikaci webového serveru jako takzvanou super smyčku, rozhodl jsem se nakonec k multitaskingovému modelu za pomoci RTOS.

6.1 ZÁKLADNÍ PARAMETRY SERVERU

Prvním požadavkem je alokace dostatečného množství paměti RAM. Alokace se provádí za pomoci správce paměti. U procesoru řady ColdFire 52233 můžeme alokovat zásobníky celkem pro čtyři úlohy. Z toho tedy vyplývá, že náš server může „najednou“ obsloužit až čtyři žádosti. Schéma činnosti naleznete na obrázku č. 9. Celý webový server se nachází v souboru *server_http.c* a jeho hlavičkovém souboru *server_http.h*. Následující popis jednotlivých funkcí a způsobu práce tohoto serveru bych nejprve začal od volání spuštění úlohy serveru. Její navázání na stack a všechny předchozí inicializace začínající v souboru *main.c*, bych si rád nechal až na závěr.

6.2 VYTVOŘENÍ ÚLOHY SERVERU

Úlohu serveru vytvoříme za pomocí volání funkce *vytvorit_ulohu_serveru* volané ve funkci *create_apptasks*, která je definovaná v souboru *tk_misc.c*. Hlavní částí vytvoření úlohy je volání funkce *TK_NEWTASK* s adresou objektu *objekt_http_uloha*. Ve skutečnosti se jedná o makro definované v souboru *osport.h*, které volá funkci RTOS API rozhraní *tk_new()*. Při tomto postupu volání makra je nová úloha vložena přímo po aktuálně probíhající úloze. Parametr makra *TK_NEWTASK* je *objekt_http_uloha*. Jedná se o strukturu typu *inet_taskinfo* deklarované v hlavičkové souboru *osport.h*. Struktura *objekt_http_uloha* obsahuje jméno struktury úlohy vytvářející soket, na kterém aplikace naslouchá za pomocí makra *TK_ENTRY*.

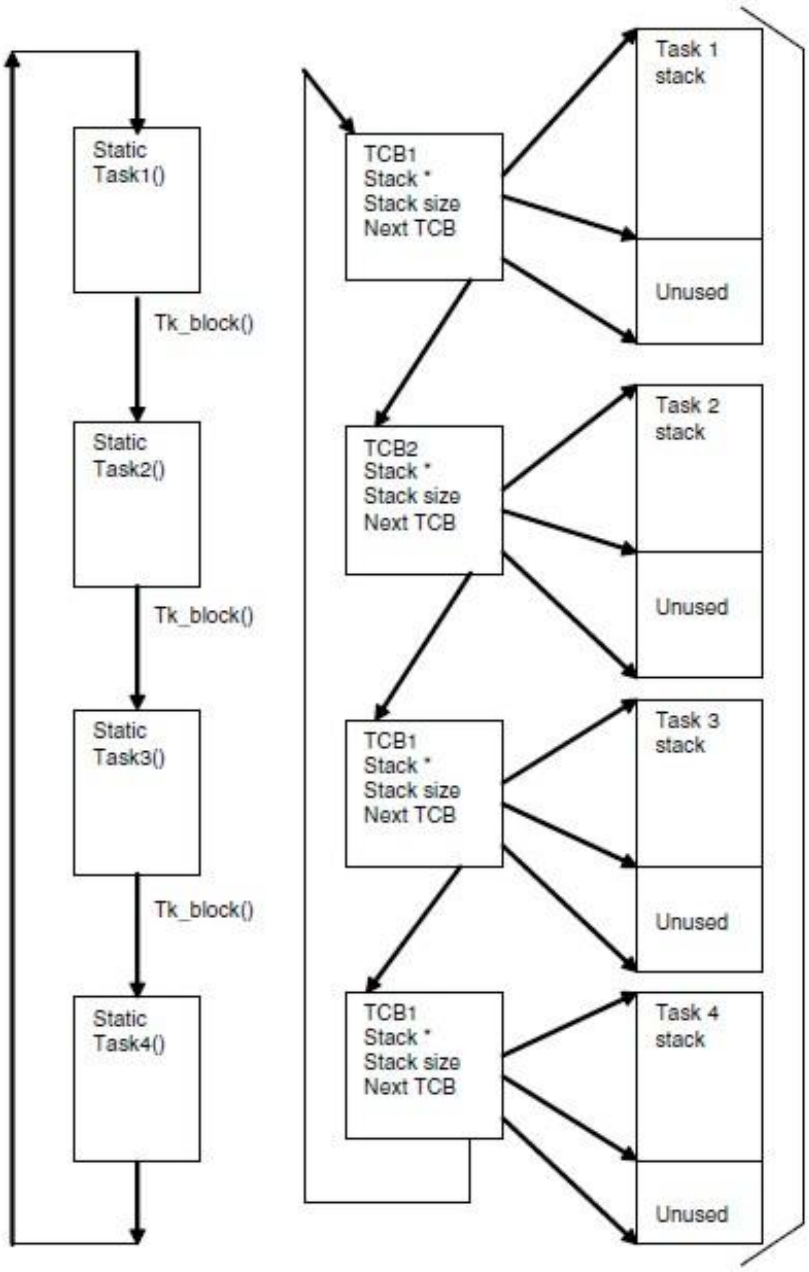
6.3 VYTVOŘENÍ NASLOUCHACÍHO SOKETU A CYKLICKÁ KONTROLA STAVU SOKETU

Výše uvedené činnosti obstarává výše zmíněné makro *TK_ENTRY*, jehož parametrem je ukazatel *tk_serverhttp* na funkci. Jako první proběhne kontrola proměnné *iniche_net_ready*, zda je RTOS připraven. Pokud není, zavoláním makra *TK_SLEEP(1)* systém uspíme. Tento cyklus se opakuje, dokud není celý systém úplně připraven.

Následuje proces inicializace, kde nejprve zneplatníme všechny relace a pomocí funkce *msring_init* vytvoříme frontu zpráv pro MINI_TCP soketové rozhraní. Naslouchací soket vytvoříme funkcí *m_listen*. Mezi jejími parametry je číslo portu, na kterém bude aplikace naslouchat. Standardní webové prohlížeče komunikují se serverem na portu 80 a tento není výjimkou. Dalším parametrem funkce je adresa funkce zpětného volání s názvem *funkce_callback*. Ta má za úkol při detekování spojení buď přidat spojovací zprávu do fronty zpráv, v případě úspěšného otevření soketu, nebo v případě neúspěšného otevření soketu relaci ukončit. Nakonec funkce zpětného volání „probudí“ úlohy serveru.

Po této počáteční inicializaci se program dostává do nekonečné smyčky, ve které probíhají dvě hlavní části. První část kontroluje, zda nedošlo ke spojení na naslouchacím soketu a následně kontroluje pomocí *msring_del*, zda nežádá klientův soket o přístup. Funkce *msring_del* si tento soket vybírá přímo z bufferu. Nyní přejde program do smyčky, která postupně prohlíží všechny relace a hledá mezi nimi volnou, kterou by mohla použít. V případě nalezení volné relace je označen status této relace na *STAV_SERVERU_HTTP_CEKA_HLAVICKU* a přijatý soket je přiřazen relaci. Mezi další parametry relace patří parametr *platne*, již dříve použitý pro kontrolu, zda je nějaká relace volná. Parametr *zustane_spojeno* má přiřazenu hodnotou 100, což znamená, že spojení se ihned po prvním obsloužení neukončí, ale po dobu přibližně 10 sekund očekávat další požadavky. Cyklus hledání volné relace se ukončí a spustí se druhá část programu, kterou je postupné prohledání relací a v případě nalezení platné relace dojde k jejímu zpracování pomocí funkce *zpracovani_pozadavku*.

Obr. 9. *Struktura modelu RTOS v porovnání se strukturou statické úlohy*



6.4 ZPRACOVÁNÍ POŽADAVKŮ

Funkce *zpracovani_pozadavku* přijímá jako svůj parametr číslo relace, kterou má zpracovat podle statusu relace.

Celkem se jedná o tři stavy, které mohou nastat.

- *STAV_SERVERU_HTTP_CEKA_HLAVICKU*
  Zavolá funkci *cteni_http_hlavicky*.
- *STAV_SERVERU_HTTP_POSLAT_SOUBOR*
  Zavolá funkci *odeslat_soubor*.
- *STAV_SERVERU_HTTP_UKONCIT*
  Zneplatní relaci a ukončí pomocí funkce *m_close*.

6.5 ČTENÍ HTTP HLAVIČKY

Zajišťuje přečtení, dekódování hlavičky a volání funkce pro její zpracování. Příjem TCP dat obstarává funkce *m_recv*. Parametry jsou soket dané relace, buffer pro dočasné uložení zpracovávaných dat a velikost přijímacího bufferu. Jako návratovou hodnotu vrací velikost přijatých dat. Zde je také dobré místo pro dekrementaci proměnné *zustane_spojeno*.
V bufferu tedy máme uloženu hlavičku soketu. Můžeme tedy ihned z prvního písmene rozpoznat, o jaký typ se jedná. Pro jistotu zkontrolujeme i zbylé znaky a spustíme zpracování hlavičky funkcí *zpracovani_hlavicky*. Sice program rozpozná dva druhy hlaviček a spustí jejich zpracování, ale následující kód pokračuje jenom hlavičkou s metodou *GET*.

6.6 ZPRACOVÁNÍ HLAVIČKY

Tato funkce získá jméno požadovaného souboru a načte hodnoty z A/D převodníků a binárních vstupů ovládaných tlačítky SW1 a SW2.
Jméno souboru získáme, stejně jako v předchozím případu metodu, z bufferu čtením dalších znaků. Požadavek na soubor odeslaný ze strany klienta vypadá následovně: *GET/jméno_souboru.htm HTTP/1.1*. V cyklu najdeme znak \ po němž následuje jméno souboru i s příponou, zakončené mezerou. Nyní již máme požadované jméno souboru a můžeme přejít k načtení binárních stavů a hodnot z A/D převodníků. Po něm následuje kontrola, zda nedošlo k požadavku o aktivaci nebo deaktivaci LED diod. LED diody jsou aktivovány a deaktivovány cyklicky, to znamená, že když

zašleme požadavek na konkrétní diodu, bude při prvním požadavku aktivována a při druhém deaktivována. Diody mají přiřazená čísla 1 až 4, podle vývojové desky, na které celý projekt zkouším. Požadavek na změnu stavu LED diody musí být v následujícím tvaru: *jméno_souboru.přípona**?led=**číslo_LED_diody*. Maximální délka jména souboru včetně otazníku a návěští je 30 znaků. Při správném zadání kódového návěští *led=* se změní stav diody při udaném čísle 1 až 4. Pokud za návěští *led=* zadáme jiné číslo, všechny diody zhasnou.

Pokračujeme nejdůležitější částí této funkce a tou je nalezení webové stránky podle požadovaného jména souboru. Webové stránky jsou uloženy v samostatném souboru s názvem *obsah_http_serveru.c*. Jejich názvy se nacházejí v poli s názvem *jmena_ulozenych_souboru*. Kontrola shodnosti názvů probíhá tak, že program kontroluje, až do velikosti jména požadovaného souboru, jména uložených souborů. Pokud se jména shodují, bude na další pozici znaku nula. Když tedy jméno požadované stránky nalezne v seznamu webových stránek, které server obsahuje, nastaví dané relaci její parametry na hledaný soubor.

Parametry relace jsou následující:

- ***http_relace_serveru[relace].typ_souboru***
  Určuje příponu souboru, kvůli dalšímu určování, zda může soubor obsahovat požadavek na dynamicky zobrazovaná data. (Hodnoty binárních vstupů a hodnoty z A/D převodníků)
- ***http_relace_serveru[relace].velikost_souboru***
  Délka souboru v bajtech.
- ***http_relace_serveru[relace].ukazatel_souboru***
  Ukazatel na název proměnné obsahující data požadovaného souboru.
- ***http_relace_serveru[relace].index_souboru***
  Do tohoto parametru ukládáme počet již přečtených bajtů u souborů, které nelze odeslat najednou.
- ***http_relace_serveru[relace].status***
  Status, kterým je řízen běh celé aplikace. V případě nastavování souboru určeného k odeslání, je nutné použít hodnotu:
  STAV_SERVERU_HTTP_POSLAT_SOUBOR

Pokud server nenalezne ve svém seznamu jméno požadovaného souboru, nastaví také relaci, ale se stránkou zobrazující nápis, že nebyla stránka požadovaného jména v serveru nalezena.
V aplikaci je použito dvou způsobů ukládání webových stránek. První je webová stránka se sdělením, že soubor nebyl nalezen. Ta se nachází v proměnné *stranka_nenalezena*, uložena nekompilovaná a její kompilace se provádí při kompilaci programu. Tento přístup by byl možný i pro ostatní stránky, ale vzhledem k jejich větší velikosti je snazší způsob uložit do proměnné stránku již převedenou do hexadecimálního kódu.

6.7 ODESLÁNÍ POŽADOVANÉHO SOUBORU

Ve funkci *odeslat_soubor*, jejímž parametrem je relace, kterou obsluhujeme, probíhá kontrola typu souboru, nahrazení návěští a samotné odeslání dat.
První je tedy kontrola typu souboru. Ta slouží k tomu, aby aplikace prohledávala soubory s příponou *html* a hledala v nich návěští ve tvaru: ***~číslo_proměnné;***
Nejprve se tedy za pomoci funkce *nacti_soubor* načtou data do pomocné proměnné *pomocna_data_k_poslani* o velikosti definované v konstantě *MAX_BYTES_TO_SEND*. Voláním funkce *nahrad_daty* je potom nalezené návěští nahrazeno příslušnou hodnotou buď z binárních vstupů nebo A/D převodníku.
Hledání návěští v souboru probíhá dvěma způsoby podle to, zda je nutné soubor rozdělit nebo je ho možné poslat najednou. Tato je nutné kvůli tomu, aby nedošlo v rozdělovaném souboru k rozdělení v místě návěští. Jeho zbytky by se jinak později zobrazovaly jako nesmyslné znaky na webové stránce.
Data připravená k odeslání a doplněná patřičnými dynamickými hodnotami jsou odeslána za pomoci funkce *m_send*. Následně je ošetřen stav, kdyby náhodou nedošlo k odeslání dat.
Pokud nejsou k odeslání žádná data, zkontroluje aplikace relaci, zda má zůstat spojena a změní její status na *STAV_SERVERU_HTTP_CEKA_HLAVICKU*. Jestliže vypršela doba určená ke spojení je status relace změněn na *STAV_SERVERU_HTTP_UKONCIT*.

6.8 FUNKCE PRO NAČTENÍ SOUBORU

Jak jsem již dříve zmiňoval, je velikost dat odesílaných v jednom paketu omezena. Její maximální hodnotu určuje konstanta *MAX_BYTES_TO_SEND*. Ve funkci zajišťující správné načtení souboru *nacti_soubor* je tato konstanta přiřazena do proměnné *max_precist_bajtu* kvůli tomu, aby bylo možné tuto hodnotu snížit pro případ, že v souboru již není dostatek dat pro zaplnění celého bufferu.
Nejprve tedy proběhne kontrola, zda požadovaná data k přečtení nepřesahují velikost souboru. Následně program kontroluje, není-li potřeba omezit již zmíněnou velikost načítaných dat ze souboru.
Čtení samotné probíhá tak, že nastavíme ukazatel data na ukazatel soubor v aktuální relaci. Potom posuneme začátek dat o hodnotu dat již dříve poslaných a uložených v relaci v proměnné *index_souboru*. Potom již kopírujeme znak po znaku do výstupního bufferu data, až do velikosti proměnné udávající maximální hodnotu přečtených dat. Nakonec aktualizujeme proměnnou *index_souboru* do které přičteme hodnotu aktuálně přečtených dat. Toto hodnotu funkce *nacti_soubor* také vrací a funkce *odeslat_soubor* ji přijímá jako proměnnou *delka*.

6.9 DYNAMICKÉ ZOBRAZENÍ HODNOT

Dynamické zobrazování hodnot z binárních vstupů, a nebo z A/D převodníků, je vyřešeno ve funkci *nahrad_daty*, volané po předchozí funkci *nacti_soubor*.
Návěští podle kterého program pozná, že na uvedené místo patři konkrétní hodnota z pole proměnných *html_promenna* je ve tvaru: *~číslo_hodnoty;* Program tedy postupně prohledá znaky v souboru a na místě kde nalezne znak tilda, se zastaví. Ihned po znaku tilda následuje číslo hodnoty, kterou má server vrátit. Potom nahradí tři pozice kolem znaku hodnotou z pole *html_promena* a jako index pole slouží číslo hodnoty.

6.10 UKONČENÍ RELACE SERVERU

Poslední hodnotou statusu http relace je *STAV_SERVERU_HTTP_UKONCIT*. Tu, podobně jako status určující, že má server čekat na hlavičku nebo odeslat soubor, obsluhuje funkce *zpracovani_pozadavku*. Program nejprve zkontroluje platné relace, a pokud je relace určená k ukončení platná, provede její zneplatnění. Přitom čeká na potvrzení ze strany klienta. Zavoláním funkce *m_close* otevřený soket uzavře.

6.11 NAVÁZÁNÍ PROGRAMU WEBOVÉHO SERVERU NA STACK OD INTERNICHE

Celý server začíná spuštěním hlavní funkce *main* v souboru *main.c*. Funkce *main* je již předvytvořena firmou Interniche a obsahuje všechna potřebná volání funkcí ke správnému spuštění TCP/IP stacku. Dále ve funkci *main* inicializujeme výstupy, ke kterým jsou připojené LED diody a inicializujeme také převodníky A/D. Nastavíme IP adresu, adresu brány a síťovou masku. Program po inicializaci heapu spustí funkci *netmain* s prázdným parametrem ze které pokud nedojde k neočekávané chybě, se již nevrátí. Ve funkci *netmain* je pro nás nejdůležitější funkce *create_apptasks* ve které dojde k zavolání funkce webového serveru *vytvorit_ulohu_serveru*. Stručný popis funkce TCP/IP stacku je uveden v předcházejících kapitolách. V souborech dodaných k elektronické verzi naleznete také mapu přehledu funkcí a jejich sledu volání při spouštění zásobníku.

7. OVĚŘENÍ FUNKČNOSTI WEBOVÉHO SERVERU

V průběhu vývoje aplikace jsem využíval pro kontrolu komunikace po síti LAN program fungující jako síťový monitor (Axence NetTools 3.2) a pro kontrolu dosažení jednotlivých částí programu jsem využíval funkce již implementované v RTOS a to je výpis hlášení na sériový port. To se právě ukázalo jako mnohdy jediný možný prostředek ke správnému odladění programu. Takto řešené odlaďování programu mne natolik zaujalo, že jsem se pokusil o napodobení této funkce i u jiných mikrokontrolérů, se kterými pracuji.

7.1 OTESTOVÁNÍ FUNKCÍ

Samotné odzkoušení hotové aplikace je jednoduché. Vývojový kit M52233DEMO připojíme k PC pomocí USB kabelu, což nám zabezpečí jeho napájení a dále pomocí patch kabelu připojíme vývojový kit do sítě LAN. IP adresa serveru je nastavena na hodnotu *192.168.0.100*. Po zadání této adresy do webového prohlížeče (doporučuji Internet Explorer), se nám zobrazí stránka se sdělením, že nebyla požadovaná stránka nalezena. Neimplementoval jsem funkci, která by po neudání jména žádané stránky zobrazila implicitně stránku se jménem *index.htm* nebo *default.htm*. To samé se stane, pokud nenajde server ve svém vnitřním seznamu název stránky totožný se stránkou požadovanou.
U hodnot z A/D převodníků a binárních vstupů je nutné pro aktualizaci hodnot tuto stránku znovu načíst. Nedochází tedy k jejímu automatickému obnovování.
Výstupy k nimž jsou připojeny LED diody, jsou ovládány pomocí tlačítek ve webovém formuláři a jejich aktivace se projeví ihned po stisknutí příslušného tlačítka.

8. ZÁVĚR

Téma této diplomové práce zní: „Implementace jednoduchého webového serveru do mikrokontroléru ColdFire MCF 52233“. Abychom mohli dosáhnout tohoto cíle, je potřeba se nejprve seznámit se základními pojmy a strukturou celého problému. Proto začíná diplomová práce pojednáním o samotném mikrokontroléru, jeho možnostech a přehledem jeho použití. Mezi důležité části bych určitě zařadil přehled jeho vnitřních modulů. Například modul převodníků A/D bude v programu použit pro získávání dynamických dat. A díky implementovanému akcelerometru, který je na tyto vstupy připojen, můžeme při znovunačtení webové stánky na ní odečítat hodnoty z tohoto senzoru.

V další kapitole je vysvětlen princip na jakém funguje protokol http a jsou zde popsány jeho základní možnosti i hlavní rozdíly mezi jednotlivými verzemi. Popis je doplněn příkladem, jak takováto komunikace probíhá. Pokud použijete program, kterým můžete monitorovat komunikování na síti a odesílat data, jako je třeba program Axence NetTools 3.2, můžete si rovnou podle uvedeného příkladu tuto komunikaci vyzkoušet. Když jsem v tomto programu zadal do pole send příkaz *GET/index.htm HTTP/1.1,* zobrazila se mi přímo v poli přijímaných dat odpověď od serveru ve formě kódu požadované stránky.

Po úvodu do protokolu http následuje kapitola nazvaná ColdFire TCP/IP stack. První části této kapitoly jsou věnovány popisu referenčního OSI modelu, TCP protokolu a prezenční vrstvě. Další části jsou už zaměřené přímo na stack a vysvětlení API funkcí RTOS, které jsou v navazující kapitole předvedeny do jednoduchého příkladu. Na kterém jsou demonstrovány základní principy komunikace pomocí stacku od Interniche. Považoval jsem tento příklad za důležitý, protože i když tyto funkce nalezneme v kapitole věnované popisu principu práce implementovaného webového serveru, tak si myslím, že uvedený příklad poskytne čtenáři lepší přehled o struktuře celé aplikace.

A pro úplné porozumění fungování aplikace slouží kapitola „Vlastní popis webového serveru“. Ta je v podstatě sestavena krok po kroku tak, jak aplikace webového serveru pracuje a vysvětluje jednotlivé části programu a jejich smysl. Ještě

podrobnější vysvětlivky jsem se snažil doplnit do samotného programu jako jeho komentáře.

Program realizující funkci webové serveru jsem vyzkoušel na vývojové desce od výrobce mikrokontroléru a vše fungovalo, tak jak má. Jenom jedna věc mne překvapila a tou byla teplota procesoru při práci. Pro jistotu jsem tedy vývojový kit doplnil malým pasivním chladičem.

Jako výbornou bych označil práci ve vývojovém prostředí CodeWarrior. Po zkušenostech s mikrokontrolérem ColdFire jsem si objednal i jinou vývojovou desku s osmibitovým mikrokontrolérem a plánuji kompletní přechod na tyto mikrokontroléry.

Toto téma diplomové práce jsem si vybral kvůli tomu, že jsem neměl žádné předchozí zkušenosti s použitím mikrokontrolérů na síti ethernet a již několikrát jsem pracoval na zařízení, kde by bylo jejich použití vítaným vylepšením. Bude ale nutné pro jejich použití aplikaci webového serveru doplnit o vhodné zabezpečení. Jako nutné bych v tomto případě viděl doplnit veškeré ovládání a snímání hodnot o takzvané časové značky, po jejichž době uplynutí by nebyl již příkaz platný. To kvůli tomu, aby jednoduše nemohlo dojít k tomu, že do webového serveru dorazí, ať už jakkoli, zpožděný paket. Zařízení by jej vyhodnotilo jako platný příkaz a provedlo jemu patřičné úkony, což by vedlo k nechtěné reakci.

V úvahu také připadá zabezpečení proti záměrnému poškození. Tuto ochranu bych ale řešil na jiné úrovni, vzhledem k výkonovým možnostem celého zařízení.

Když se závěrem zamyslím nad možností uplatnění této aplikace, dostávám se k nepřebernému množství zařízení fungujících jako terminály, dálkové ovládání strojů a budov, zabezpečovací technika, univerzální měřící moduly, vzdálený monitoring. Toto vše přináší výhoda zařízení připojeného ethernetovým rozhraním k síti LAN a následně k internetu.

9. SEZNAM POUŽITÉ LITERATURY

[1] Freescale, MCF52235 ColdFire® Integrated Microcontroller Reference Manual [online]. 09/2007, revision 5, [cit. 09/2007]. http://www.freescale.com/files/32bit/doc/ref_manual/MCF52235RM.pdf?fsrch=1

[2] Gregori Eric, ColdFire Lite HTTP Server [online]. 04/2007, revision 0, [cit. 04/2007]. http://www.freescale.com/files/microcontrollers/doc/app_note/AN3455.pdf?fsrch=1

[3] Gregori Eric, Advanced ColdFire TCP/IP Clients [online]. 09/2007, revision 0, [cit. 09/2007]. http://www.freescale.com/files/32bit/doc/app_note/AN3518.pdf?fsrch=1

[4] Gregori Eric, Small Footprint ColdFire TCP/IP Stack [online].]. 09/2007, revision 0, [cit. 09/2007]. http://www.freescale.com/files/32bit/doc/app_note/AN3507.pdf?fsrch=1

[5] Gregori Eric, ColdFire TCP/UDP/IP Stack and RTOS [online].]. 06/2007, revision 0, [cit. 06/2007]. http://www.freescale.com/files/32bit/doc/app_note/AN3470.pdf?fsrch=1

[6] KOSEK, Jiří. Html – tvorba dokonalých stránek: podrobný průvodce. Ilustroval Ondřej Tůma. 1. vyd. Praha: Grada, 1998. 291 s. ISBN 80-7169-608-0.

[7] DOSTÁLEK, Libor. Velký průvodce protokoly TCP/IP – Bezpečnost. 1.vyd. Praha, Computer Press, 2001.565 s. ISBN 80-7226-513-X

[8] DOSTÁLEK, L. – KABELOVÁ, A. Velký průvodce protokoly TCP/IP a systémem DNS. 1.vyd. Praha, Computer Press 2000. 426 s. ISBN 80-7226-323-4

[9] BIGELOW, J. S. Mistrovství v počítačových sítích.1.vyd. Computer Press 2004. 990 s. ISBN 80-251-0178-0

[10] BURKHARD, Mann. C pro mikrokontroléry µC & praxe. 1.vyd. Praha, BEN – technická literatura 2003. 275 s. ISBN 80-7300-077-6

[11] PRATA, S. Mistrovství v C++. 1.vyd. Computer Press 2001. 966 s. ISBN 80-7226-339-0

10. SEZNAM POUŽITÝCH ZKRATEK

RICS - zmenšená sada programovacích instrukcí

SRAM – statická paměť s náhodným přístupem

kbyt – kilo bajt (1024 bajtů)

DMA - přímí přístup do paměti (direct memory access)

FlexCAN – je síťová architektura v embedded zařízeních, která rozšiřující možnosti CAN

CAN – řízená plošná síť

I2C – meziobvodová komunikace – označení multi masterové sériové sítě používané především v integrovaných obvodech

UART - univerzální asynchronní přijímač/vysílač

SPI – sériová periferní síť – taktéž používána převážně u integrovaných obvodů

EMAC - snížená instrukční programovací sada s rozšířeným násobícím akumulátorem

CAU - kryptologická urychlovací jednotka

FEC - fast ethernet kontrolér

ePHY – přijímač ethernetu na čipu

ADC, A/D - analogově-digitální převodník

QSPI - moduly pro řazené rozhraní sériových periférií

DES, 3DES, AES, MD5, SHA-1 – kryptovací algoritmy

http - hypertextový přenosový protokol

www - word wide web

ASCII – americké standardní kódy pro výměnu informací

RTOS - multitaskingový operační systémem

TCP – řízený přenosový protokol

OSI – takzvaný OSI model, je otevřený referenční model systému pro vzájemnou komunikaci

ARP – protokol pro rozlišení adres

IP – internetový protokol

ICMP – internetový protokol kontrolní zprávy

UDP – datagramový protokol

TCP – přenos kontrolující protokol

DHCP – protokol dynamické konfigurace hosta

BOOTP - bootstrap protocol

TFTP – jednoduchý protokol pro přenos souborů

IPv4, IPv6 – internetový protokol verze 4 a 6

SDK - pro vývojáře síťových

PPP – spojení z bodu do bodu

PPPoE – spojení z bodu do bodu přes ethernet

SNMP – jednoduchý síťový ovládací protokol

DNS – protokol překladu doménových jmen

ACK – potvrzení příjmu zprávy

API – přístupové rozhraní k uživatelským funkcím

LED – světlo emitující dioda

MIME – hlavička multi účelového internetového poštovního rozšíření

11. SEZNAM PŘÍLOH

Veškeré přílohy se nalézají na přiloženém CD.

Jednotlivá jména složek jsou:

- SERVER – tato složka obsahuje kompletní program webového serveru, včetně TCP/IP stacku od Interniche
- Diplomová práce – elektronická verze diplomové práce.
- MAPA – obrázek přehledu mapy funkcí při inicializaci stacku