ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Implementace operačního systému Linux pro procesor NIOS II

vedoucí práce: Ing. Petr Burian
autor: František Remiáš

2012

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. František REMIÁŠ**
Osobní číslo:	**E09N0206P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Telekomunikační a multimediální systémy**
Název tématu:	**Implementace operačního systému Linux pro procesor NIOS II**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování:

1. Seznamte se se softwarovým procesorem NIOS II a vývojovým kitem Altera DE2-70.
2. Analyzujte možnosti nasazení operačního systému Linux na platformy založené na procesoru NIOS II.
3. Vytvořte vhodnou distribuci operačního systému Linux pro procesor NIOS II a periférie obsažené na vývojovém kitu.
4. Na jednoduchých příkladech prezentujte přístup uživatelských aplikací k periferiím procesoru.
5. Celý postup důkladně popište.

Rozsah grafických prací: **podle doporučení vedoucího**

Rozsah pracovní zprávy: **30 - 40 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

Student si vhodnou literaturu vyhledá v dostupných pramenech podle doporučení vedoucího práce.

Vedoucí diplomové práce: **Ing. Petr Burian**
Regionální inovační centrum elektrotechniky

Konzultant diplomové práce: **Ing. Petr Burian**
Regionální inovační centrum elektrotechniky

Datum zadání diplomové práce: **18. října 2010**

Termín odevzdání diplomové práce: **11. května 2012**

Doc. Ing. Jiří Hammerbauer, Ph.D.
děkan

L.S.

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 17. října 2011

Implementace operačního systému Linux pro procesor NIOS II

Anotace

Předkládaná diplomová práce se zabývá implementací operačního systému Linux na procesor NIOS II. Tato diplomová práce analyzuje dostupná komerční i volně šiřitelná řešení distribucí operačního systému Linux, která lze pro tento softwarový procesor použít. Diplomová práce dále popisuje možnosti volně šiřitelné distribuce NIOS II Linux Communinty Edition a zabývá se jejími úpravami. Na demonstračních aplikacích ukazuje přístup k periferiím z operačního systému Linux. Pro tyto účely byl využit vývojový kit DE2-70 od společnosti Terasic. Diplomová práce se zabývá popisem systému s podporou jednotky správy paměti (MMU) i systémem bez podpory MMU.

Klíčová slova

NIOS II, embedded Linux, úpravy distribuce uClinux, přístup k periferiím z prostředí operačního systému Linux, výkon procesoru NIOS II v operačním systému Linux, NIOS II Linux, operační systém bez podpory MMU

Implemening of the Linux operating system to the NIOS II processor

Abstract

This master thesis deals with the implementation of the Linux operating system on the NIOS II processor. The thesis analyzes available commercial and open source solutions of Linux distributions, which can be used for a NIOS II soft-core processor. The master thesis also describes features of the open source distribution NIOS II Linux Community Edition and deals with modifications of this distribution. This thesis also shows on demonstrative applications how to control periphery from the Linux operating system. For these purposes has been used the development kit DE2-70. This master thesis deals with a description of systems with and without memory management unit (MMU) support.

Key words

NIOS II, embedded Linux, modification of uClinux distribution, access to hardware from Linux operating system environment, benchmark tests of NIOS II processor in Linux, NIOS II Linux, operating system without MMU support

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 10.5.2012 František Remiáš

……..

Obsah

Seznam použitých symbolů a zkratek

ALU	Arithmetic Logic Unit (aritmeticko-logická jednotka)
API	Application Programming Interface (rozhraní pro programování aplikací)
BSD	Barkeley Software Distribution (obchodní organizace při University of California, Berekley)
CFI	Common Flash Interface (rozhraní pro připojení Flash paměti)
CGI	Common Gateway Interface (protocol pro propojení aplikací s webovým serverem)
CPU	Central Processing Unit (centrální procesorová jednotka)
DAC	Digital Analog Converter (digitálně analogový převodník)
DHCP	Dynamic Host Configuration Protocol (dynamické přidělování IP adres)
DSP	Digital Signaling Processor (digitální signálový prcesor)
EIC	External Interrupt Controller (externí řadič přerušení)
EOF	End Of File (konec souboru)
EPCS	Serial Enhanced Configuration Devices (sériové rozšířitelné konfigurovatelné zařízení)
FAT	File Allocation Table (tabluka rozmístění souborů)
FPGA	Field Programmable Gate Array (programovatelné hradlové pole)
FTP	File Transfer Protocol (protocol pro přenos souborů)
GCC	GNU Compiler Collection (sada překladačů GNU)
GDB	GNU Project Debugger (ladící program GNU)
GPL	General Public Licence (všeobecná veřejná license)
HID	Human Interface Device (zařízení lidského rozhraní)
HTTP	Hyper Text Transfer Protocol (internetový protokol pro výměnu hypertextových dokumentů)
I/O	Input/Output (vstup/výstup)
IRQ	Interrupt ReQuest (požadavek na přerušení)
ISR	Interrupt service (služba přerušení)
JTAG	Join Test Action Group (standard IEEE 1149.1 pro programování Flash pamětí, ladění obvodů, testování plošných spojů, atd.)
LCD	Liquid Crystal Display (displej z tekutých krystalů)
LDAT	Linux Distribution Assembly Tool (nástroj pro sestavení distribuce operačního systému Linux)
LED	Light-Emitting Diode (dioda emitující světlo)
MIPS	Million instructions per second (million instrukcí za vteřinu)
MMU	Memory Management Unit (jednotka pro správu paměti)
MPU	Memory Protection Unit (jednotka pro ochranu paměti)
NFS	Network File System (síťový souborový systém)
PCB	Printed Circuit Board (deska plošného spoje)
PIC	Position Independent Code (kód nezávislí na pozici)
PIO	Parallel Input/Output (Paralelní vstup/výstup)
RAM	Random Access Memory (paměť s náhodným přístupem)
RISC	Reduced Instruction Set Computer (počítač s redukovanou instrukční sadou)
ROM	Read-Only Memory (paměť pouze pro čtení)

RPM	RPM Package Manager (balíčkovací systém RPM)
SSH	Secure Shell (bezpečný shell)
TLB	Translation Look-aside Buffer (paměť obsahující tabulku pro překlad virtuálních adres na fyzické)
USB	Universal Serial Bus (univerzální sériová sběrnice)
VGA	Video Graphics Array (standard pro počítačovou zobrazovací techniku)
XIP	Execute In Place (přímé spuštění aplikací z jejich umístění v paměti)

Konvence

Zdrojový kód

Kompletní zdrojové kódy uvedené v této práci lze nalézt v přílohách. Zdrojové kódy, které jsou uvedeny v textu, jsou značeny tak, jak je uvedeno v následujícím příkladu:

```
1: int main()
2: {
3:  return 0;
4: }
```

Syntaxe funkcí

Názvy funkcí v textu a jejich syntaxe jsou pro lepší orientaci v textu psány následujícím stylem `void main (unsigned int var1, void *var2)`

Cesty k adresářům

Cesty k adresářům jsou uvedeny mezi značkami < >. Obvykle se jedná o umístění programů, zdrojových kódů, knihoven, atp.
Rozdílné operační systémy používají různé konvence pro značení adresářové struktury a souborů. Zatímco operační systém Windows používá lomítka zpětná „\“, systém Linux používá lomítka obyčejná „/“. Je nutno podotknout, že operační systém Linux na rozdíl od systému Windows rozeznává v cestách souborů a adresářů malá a velká písmena.

Příkazy operačního systému Linux

Příkazy operačního systému Linux jako />**ls** pro výpis obsahu adresáře, nebo />**chmod** pro změnu práv souborů budou psány tučným písmem s uvedenou značkou />, která je obvyklá pro prostředí příkazového řádku a tučným písmem. Je nutné si uvědomit, že prostředí operačního systému Linux rozeznává malá a velká písmena. Z tohoto důvodu jsou všechny příkazy psány přesně tak, jak jsou používány na příkazové řádce.

Názvy souborů

V textu se objevuje celá řada souborů. Pro jejich snadnou indentifikaci jsou v textu označeny tučným písmem. Konfigurační soubor je tak například označen **.config**

Obsah souborů a výpisy z příkazového řádku

Celé a částečné výpisy obsahu souborů a výpisy obsahu příkazové řádky jsou psány fontem `Courier, velikosti 10`. Dále je uveden příklad výpisu:

```
root:/> ls -l /dev
crw-------    1 root     root        5,   1 Jan  1 00:00 console
brw-r-----    1 root     root      179,   0 Apr 25  2012 mmcblk0
```

Názvy programů a nástrojů

Názvy programů a aplikací používaných v prostředí operačního systému Linux jako například MAKE, LS a další jsou v textu označeny KAPITÁLKAMI.

Seznam obrázků

Seznam tabulek

Seznam grafů

Úvod

Uvedená diplomová práce se zabývá možnostmi implementace operačního systému Linux na procesor se softwarovým jádrem NIOS II. Cílem této práce je analyzovat řešení a možnosti současných projektů a ukázat možnosti vývoje aplikací a ovládání periferií připojených k FPGA obvodu z prostředí operačního systému Linux. Praktická část obsahuje návod na sestavení a spuštění dostupné *open-source* distribuce operačního systému Linux na vývojové desce DE2-70 s FPGA čipem Cyclone II. Dále tuto distribuci vhodným způsobem modifikovat pro periférie obsažené na desce. Vzhledem k tomu, že k této problematice existuje velmi málo odborné literatury, poskytuje práce poskytnout souhrn informací načerpaných na zahraničních internetových fórech a diskusních skupinách, zabývajících se procesorem NIOS II a operačním systémem Linux pro embedded zařízení.

Úvodní kapitola této diplomové práce se zabývá základními vlastnostmi a možnostmi konfigurace procesoru NIOS II. Následující dvě kapitoly mají za cíl popsat vlastnosti operačního systému Linux pro embedded zařízení a základní metody a funkce pro psaní aplikací a ovladačů pro tento systém. Dále je uveden přehled dostupných komerčních i *open-source* řešení distribucí operačního systému Linux pro procesor NIOS II. Kapitola popisuje několik stávajících distribucí z hlediska řešení a obsahu software. Následující kapitola se zabývá postupem spuštění volně šiřitelné distribuce operačního systému Linux na vývojové desce DE2-70, modifikacemi této distribuce a zprovozněním některých periférií obsažených na desce. Na jednoduchých periférií ch jsou zde demonstrovány možnosti vytvoření vlastních ovladačů a aplikací. Poslední kapitola je věnována výkonnostním (*benchmark*) testům za použití algoritmů Dhrystone a Whetstone. V této kapitole jsou porovnány typy jader procesoru NIOS II a jejich různé konfigurace. Také jsou provedeny testy rychlosti zápisu a čtení paměti RAM.

Práce obsahuje přílohy s modifikacemi a zdrojovými kódy, které byly vytořeny pro účely této diplomové práce. Obsahem jsou například ovladače demonstrující ovládání periférií a jednoduché aplikace.

Celá práce je koncipována jako souhrn informací, postupů a možností nasazení operačního systému Linux na procesor NIOS II. Dále práce podrobně popisuje metody přístupu k hardware z prostředí operačního systému Linux, možnosti jeho využití a základní software, který je možno využít pro embedded systémy.

1 Softwarový procesor NIOS II

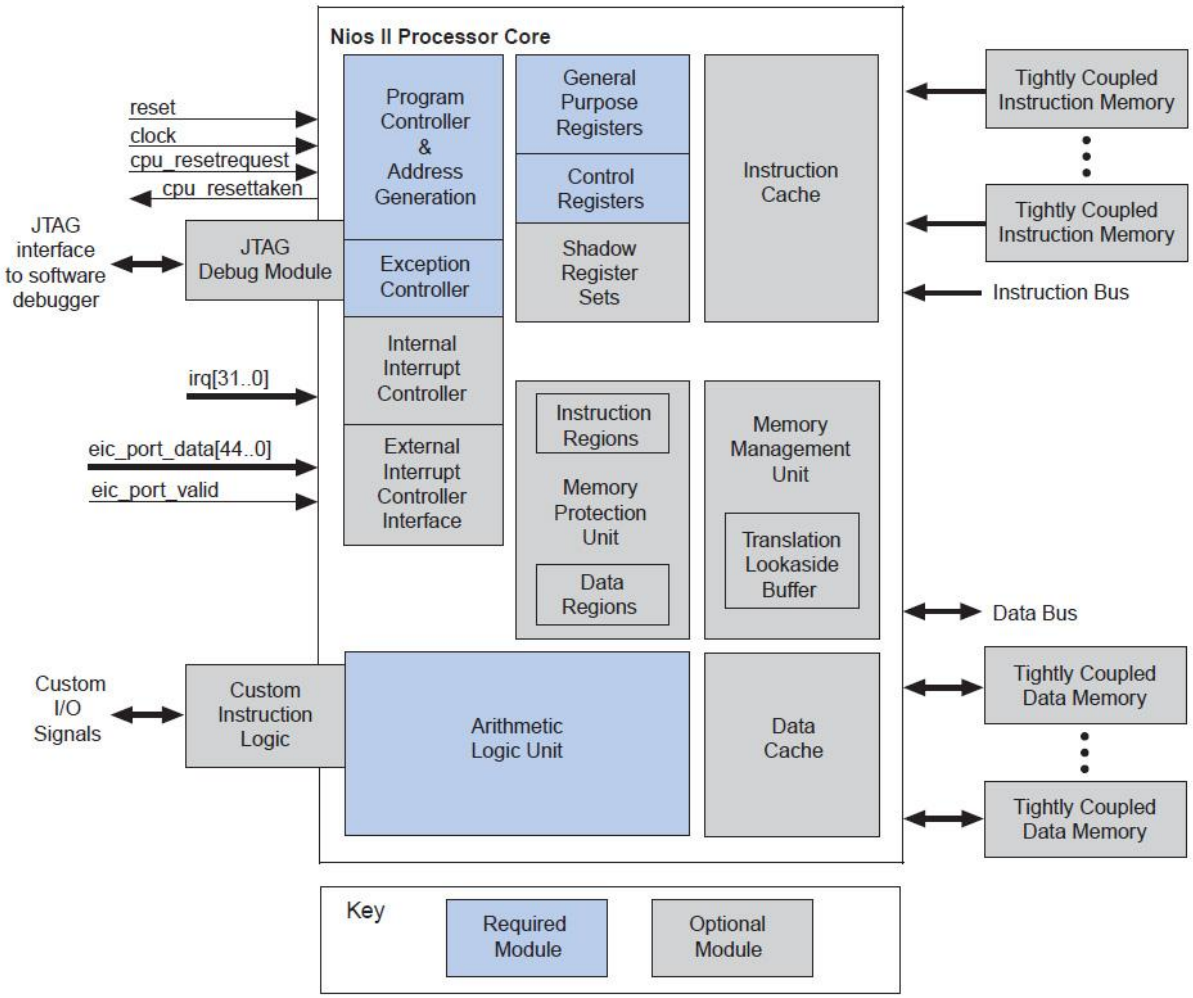
Procesor NIOS II společnosti Altera je 32bit-ový softwarový procesor založený na architektuře RISC. Procesor je plně konfigurovatelný, umožňuje řetězové zpracování a obsahuje jádro typu *soft-core*. Samo jádro procesoru NIOS II používá pouze malý počet logických elementů obvodu FPGA, což umožňuje použít spolu s ním další standardní komponenty, popř. připojit vlastní hardware. Je tak možno konfigurovat procesor, komponenty i instrukce. Pro návrh systému s jádrem NIOS II lze použít nástroje SOPC Builder nebo Qsys, které jsou součástí softwaru Quartus II of společnosti Altera [1].

1.1 Architektura procesoru NIOS II

Architektura procesoru NIOS II definuje následující funkční jednotky:

- Soubor registrů
- Aritmeticko-logickou jednotku (ALU)
- Rozhraní pro vlastní instrukční logiku
- Řadič výjimek
- Řadič vnitřního nebo vnějšího přerušení
- Instrukční sběrnici
- Datovou sběrnici
- Jednotku správy paměti (MMU)
- Jednotku pro ochranu paměti (MPU)
- Instrukční a datovou vyrovnávací paměť (cache)
- *Tightly coupled* (úzce propojenou) paměť
- Ladící JTAG modul

Funkční jednotky architektury NIOS II jsou základem instrukční sady procesoru. To však nznamená, že každá z těchto jednotek musí být implementována hardwarově. Funkční jednotka může být kromě hardwarového provedení emulována softwarově nebo úplně vynechána. Procesor NIOS II lze zkonfigurovat pro konkrétní potřeby, jako je například menší velikost jádra nebo vyšší výkon. Tato flexibilita umožňuje architekturu procesoru NIOS II přizpůsobit pro různé cílové aplikace [1].



Obr. 1.1 Blokové schéma jádra procesoru NIOS II (převzato z [1])

1.1.1 Soubor registrů

Architektura procesoru NIOS II podporuje soubor přímých registrů, které se skládají ze třiceti dvou 32-bitových registrů pro obecné použití a až třiceti dvou 32-bitových řídících registrů. Architektura podporuje režim uživatelský a režim supervizitora. Tyto režimy umožňují systémovému kódu zamezit nespolehlivými aplikacím přístup k řídícím registrům.

1.1.2 Aritmeticko-logická jednotka (ALU)

Aritmeticko-logická jednotka (ALU) pracuje s daty uloženými v registrech. Operace, vykonávaná ALU, vybere jeden nebo dva vstupy z registru a výsledek do něj zpět uloží. ALU podporuje datové operace uvedené v tab. 1.1.

Kategorie	**Popis**
Aritmetické operace	Sčítání, odčítání, násobení a dělení znaménkových a neznaménkových operandů
Relační operace	(=, !=, <, >, <=, >= u znaménkových a neznaménkových operandů
Logické operace	AND, NAND, OR, NOR a XOR
Posuny a rotace	Lze posouvat, rotovat data o 0 až 31 bitových pozic za jednu instrukci. Jednotka podporuje posun a rotaci vlevo i vpravo

Tab. 1.1 Operace podporované aritmeticko-logickou jednotkou (ALU)

1.1.3 Řadič výjimek a přerušení

Procesor NIOS II obsahuje modul pro zpracování výjimek, včetně modulu přerušení. Obsahuje také rozhraní pro externí řadič přerušení (EIC). Připojením vlastního řadiče přerušení k EIC rozhraní je tak možno urychlit zpracování přerušení ve složitém systému.

Architektura procesoru NIOS II poskytuje jednoduchý řadič pro zpracování všech typů výjimek. Každá výjimka, včetně vnitřních hardwarových přerušení, způsobí, že procesor spustí instrukce na adrese výjimky, kterou lze specifikovat v nástroji Qsys nebo SOPC Builder. Obsluha výjimky na této adrese se pokusí zjistit její příčinu a vykoná příslušnou operaci.

Procesor NIOS II podporuje 32 vnitřních hardwarových přerušení, přičemž jádro procesoru rozlišuje 32 úrovňový systém vstupních požadavků na přerušení, irq0 až irq31. Prioritu přerušení je možno nastavit softwarově.

1.1.4 Vyrovnávací paměť cache

Architektura procesoru NIOS II obsahuje datovou a instrukční vyrovnávací paměť cache. Obě tyto paměti jsou tvořeny embedded pamětí v obvodu FPGA. Cache paměť může snížit průměrný čas přístupu do paměti (např. u systémů využívajících paměť SDRAM pro program a data, která je umístěna mimo obvod FPGA). Podle velikosti jádra procesoru a požadavků na výkon lze procesor NIOS II zkonfigurovat tak, že bude obsahovat obě paměti cache, pouze jednu z nich nebo žádnou. V konfiguraci procesoru lze rovněž zvolit velikost těchto pamětí.

1.1.5 Paměť typu tightly-coupled (úzce propojená)

Tato paměť garantuje přístup s nízkou latencí, zejména pro aplikace s kritickou závislostí na výkonu. V porovnání s pamětí cache přináší paměť *tightly-coupled* následující výhody:

- Výkon přibližně stejný s pamětí cache
- Software může garantovat umístění kódu nebo dat závislých na výkonu do paměti *tight-coupled.*
- Nedochází k přetížení paměti v reálném čase, jako je načítání, ověření nebo vymazání paměti.

1.1.6 Jednotka správy paměti (MMU)

Jednotka správy paměti, neboli *Memory Management Unit* (MMU) byla dříve řešena jako samostatný obvod, dnes bývá běžně součástí procesoru. Funkce MMU spočívá v překladu virtuální adresy na fyzickou, což umožňuje ochranu paměti systému.

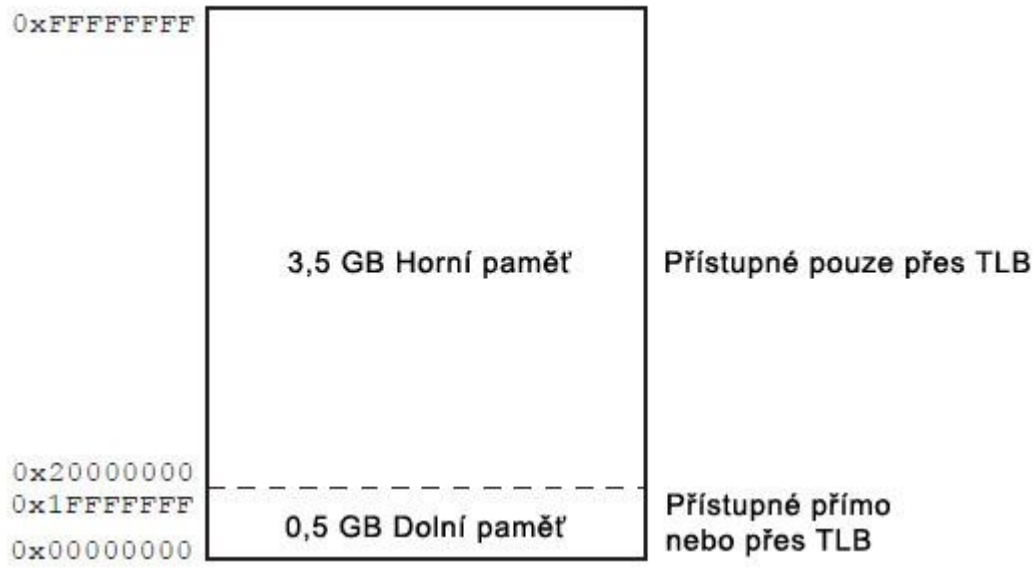
Jednotka MMU, obsažená v procesorech NIOS II, rozděluje virtuální paměť na stránky o velikostech 4KB a fyzickou paměť do bloků o stejné o velikosti. MMU pak překládá stránky virtuální paměti na fyzickou pomocí cache paměti, která se nazývá *Translation Lookaside Buffer* (TLB).

MMU umožňuje adresovat až 4GB paměti. Tento adresní prostor je rozdělen na dvě části. Horní 2GB jsou rezervovány pro operační systém, zbylý prostor je rezervován pro uživatelské procesy. Následující tabulka ukazuje rozdělení tohoto adresního prostoru.

Rezervováno pro	Rozsah adres	Uživatelský přístup
I/O	0xE0000000–0xFFFFFFFF	NE
Jádro	0xC0000000–0xDFFFFFF	NE
Jádro s MMU	0x80000000–0xBFFFFFFF	NE
Uživatel	0x00000000–0x7FFFFFFF	přes TLB

Tab. 1.2 Rozdělení adresního prostoru

Adresní prostor fyzické paměti je rozdělen na dolní a horní paměť. Spodních 512MB tohoto prostoru je označeno jako dolní a zbytek jako horní paměť viz následující obrázek. Procesor může přistupovat do dolní části paměti přímo, bez TLB.



Obr. 1.2 Přístup procesoru k paměti

Následující tabulka 1.3 srovnává výhody a nevýhody systému s a bez podpory MMU

Systém s MMU	**Systém bez MMU**
Potřeba více logických obvodů v FPGA	Potřeba méně logických obvodů v FPGA
Sdílené knihovny	Knihovny jsou pouze statické
Virtuální paměť, oddělené adresní prostory pro jednotlivé procesy	Jeden adresní prostor společný pro všechny procesy a jádro
Celkově pomalejší systém	Rychlejší
Větší velikost obraz systému	Menší velikost obrazu systému

Tab. 1.3 Porovnání výhod a nevýhod systému s jednotkou správy paměti

1.1.7 Jednotka pro ochranu paměti (MPU)

Memory Protection Unit (jednotka pro ochranu paměti) má následující funkce a vlastnosti:

- Ochranu paměti
- Až 32 instrukčních oblastí a 32 datových oblastí
- Proměnou velikost instrukční a datové oblasti
- Řízení práv pro čtení a zápis do oblasti dat
- Řízení práv pro spouštění pro oblast instrukcí
- Oblasti překrývání

MPU obsahuje podporu pro dva režimy, uživatelský a super-uživatelský. V super-uživatelském režimu běží zpravidla operační systém, nebo jádro programu. Po zažádání o přístup, může v super-uživatelském režimu běžet i libovolný software. Ve výchozím nastavení je MPU vypnutý a aktivuje se až zápisem do příslušného registru.

1.1.8 Ladící modul JTAG

Architektura procesoru NIOS II podporuje ladící modul JTAG, který umožňuje ovládat procesor vzdáleně z PC. Ladící software, běžící na vzdáleném počítači, komunikuje s ladícím modulem JTAG a poskytuje následující možnosti:

- Nahrání programu do paměti
- Spuštění a zastavení vykonávání instrukcí
- Nastavení breakpointů a watchpointů
- Analyzování registrů a paměti
- Sběr dat v reálném čase

1.1.9 Rozhraní Avalon

Rozhraní Avalon zjednodušuje návrh systému tím, že umožňuje propojení různých komponent v FPGA obvodu. Avalon definuje rozhraní, která jsou vhodná pro streamování vysokorychlostních dat, čtení a zápis registrů a ovládání zařízení, které jsou připojeny mimo hlavní obvod FPGA. Specifikace Avalon definuje následujících sedm rozhraní:

- *Avalon streaming interface* (Avalon-ST) je rozhraní, které podporuje jednosměrný tok dat, včetně multiplexovaného streamování paketů a dat DSP.

- *Avalon Memory Mapped Interface* (Avalon-MM) je paměťově mapované rozhraní pro signály typu čtení a zápis. Toto rozhraní je typické pro propojení master-slave.
- *Avalon Conduit Interface* je rozhraní pro individuální signály, nebo skupiny signálů, které nezapadají do ostatních druhů sběrnic Avalon.
- *Avalon Tri-State Conduit Interface* (Avalon-TC) je rozhraní pro připojení periférií, které jsou mimo vlastní FPGA obvod. Několik periférií může sdílet piny pomocí multiplexování signálů, což vede ke snížení počtu pinů FPGA obvodu a počtu spojů na PCB.
- *Avalon Interrupet Interface* je rozhraní, které umožňuje komponentám signalizovat přerušení.
- *Avalon Clock Interface* je rozhraní, které řídí a přijímá hodinové signály. Všechny rozhraní Avalon jsou synchronní.
- *Avalon Reset Interface* je rozhraní, které poskytuje připojení k resetovacímu signálu.

1.2 Druhy jader procesoru NIOS II

Rodina softwarových procesorů NIOS II se v současné době skládá ze tří různých jader procesoru, je tak možné zvolit vhodný poměr mezi cenou a výkonem [2].

1.2.1 Jádro NIOS II/e (economy)

Toto jádro bylo navrhnuto k dosažení malé velikosti a využití logických elementů, což vede k některým funkčním omezením. Jádro je tak určeno především pro použití tam, kde není požadavek na vysoký výkon a naopak je požadovaná nízká cena.
Hlavní rysy jádra:

- Až 2 GB externího adresového prostoru
- Ladící modul JTAG
- Kompletní systém v méně než 700 logických elementech
- Volitelné příslušenství pro ladění
- Až 256 vlastních instrukcí
- Je zdarma (není nutná žádná licence)

1.2.2 Jádro NIOS II/s (standard)

Jádro NIOS II/s bylo navrhnuto jako kompromis mezi výkonem a cenou. Obsahuje další funkce a vyšší výkon s cílem implementace bloku DSP. Jádro dále obsahuje hardwarové obvody násobení.

Hlavní rysy jádra:

- Vyrovnávací paměť pro instrukce
- Až 2 GB externího adresového prostoru
- Možnost tzv. *tightly coupled* (úzce propojené) paměti pro instrukce ke snížení latence
- Ladící modul JTAG
- 5-ti stupňovou pipeline
- Statická predikce větvení
- Volitelně hardwarové násobení, dělení a posun
- Volitelné příslušenství pro ladění, zahrnující hardwarové body přerušení, spouštěč a sledování v reálném čase
- Až 256 vlastních instrukcí

1.2.3 Jádro NIOS II/f (fast)

Výsledkem návrhu rychlého jádra je mnoho konfiguračních voleb, které umožňují vyladit procesor pro nejvyšší možný výkon. Je proto vhodný pro aplikace vyžadující vysoký výkon a zpracovávající velké množství kódu a dat, jako je běh plnohodnotných operačních systémů.

Hlavní rysy jádra:

- Jednotka pro správu paměti (MMU)
- Jednotka pro ochranu paměti (MPU)
- Kontrolér externího vektoru přerušení
- Rozšířená podpora přerušení
- Samostatné instrukce a vyrovnávací paměť dat (512 B až 64 KB)
- Až 2 GB externího adresového prostoru
- Možnost tzv. *tightly coupled* (úzce propojené) paměti pro instrukce ke snížení latence
- Ladící modul JTAG
- 6-ti stupňovou pipeline

- Dynamická predikce větvení
- Volitelně hardwarové násobení, dělení a posun
- Volitelné příslušenství pro ladění, zahrnující hardwarové body přerušení, spouštěč a sledování v reálném čase
- Až 256 vlastních instrukcí a neomezený počet hardwarových akcelerátorů

1.2.4 Výkon softwarového procesoru NIOS II

Výkon softwarového procesoru může být velmi závislý na použité verzi porgramu Quartus II, verzi procesoru NIOS II a cílovém zařízení. Výkon může být dále ovlivněn změnami v logickém návrhu systému a počtu použitých logických elementů [3].

Následující tabulky udávají maximální frekvenci a výkon softwareového procesoru NIOS II na různých FPGA obvodech. Testy byly prováděny na systému obsahujícím tyto komponenty:

- NIOS II procesor s ladícím modulem JTAG
- JTAG UART
- 64 KB *on-chip* paměti
- Avalon MM pipeline
- Časovač

Výkon procesoru byl měřen testem Dhrystone ver. 2.1, který je používán jako průmyslový standard pro měření výkonu procesorů. Testu Dhrystone je dále věnována kapitola 7.2.1.

		f_{max} [MHz]		
Rodina zařízení	**Použité zařízení**	**NIOS II/f**	**NIOS II/s**	**NIOS II/e**
Stratix® V	5SGXMA3H2F35C2	250	240	320
Stratix IV	EP4SGX230HF35C2	230	240	300
Stratix III	EP3SL150F1152C2	290	230	340
Stratix II	EP2S60F1020C3	220	170	285
Stratix	EP1S80F1020C5	150	130	170
HardCopy® IV	HC4E35FF1152	305	285	290
HardCopy III	HC322FF1152	230	220	210
HardCopy II	HC230F1020C	200	200	320
HardCopy Stratix	EP1S80F1020C5_HC	150	130	175
Cyclone IV GX	EP4CGX30CF19C6	150	120	175
Cyclone III LS	EP3CLS70F484C7	135	100	140
Cyclone III	EP3C40F324C6	175	145	215
Cyclone II	EP2C20F484C6	175	145	215
Cyclone	EP1C20F400C6	135	120	175
Arria® II	EP2AGX95DF25C4	230	200	270
Arria GX	EP1AGX60CF484C6	140	100	150

Tab. 1.4 Maximální frekvence jader procesoru NIOS II pro konkrétní FPGA obvody (převzato z [3]).

		DMIPS		
Rodina zařízení	**Použité zařízení**	**NIOS II/f**	**NIOS II/s**	**NIOS II/e**
Stratix® V	5SGXMA3H2F35C2	283	154	49
Stratix IV	EP4SGX230HF35C2	260	154	48
Stratix III	EP3SL150F1152C2	340	140	48
Stratix II	EP2S60F1020C3	250	110	45
Stratix	EP1S80F1020C5	170	80	27
HardCopy® IV	HC4E35FF1152	345	180	45
HardCopy III	HC322FF1152	260	140	30
HardCopy II	HC230F1020C	230	130	50
HardCopy Stratix	EP1S80F1020C5_HC	165	85	57
Cyclone IV GX	EP4CGX30CF19C6	170	77	27
Cyclone III LS	EP3CLS70F484C7	153	64	22
Cyclone III	EP3C40F324C6	195	90	30
Cyclone II	EP2C20F484C6	145	55	18
Cyclone	EP1C20F400C6	130	52	17
Arria® II	EP2AGX95DF25C4	260	128	42
Arria GX	EP1AGX60CF484C6	150	65	25

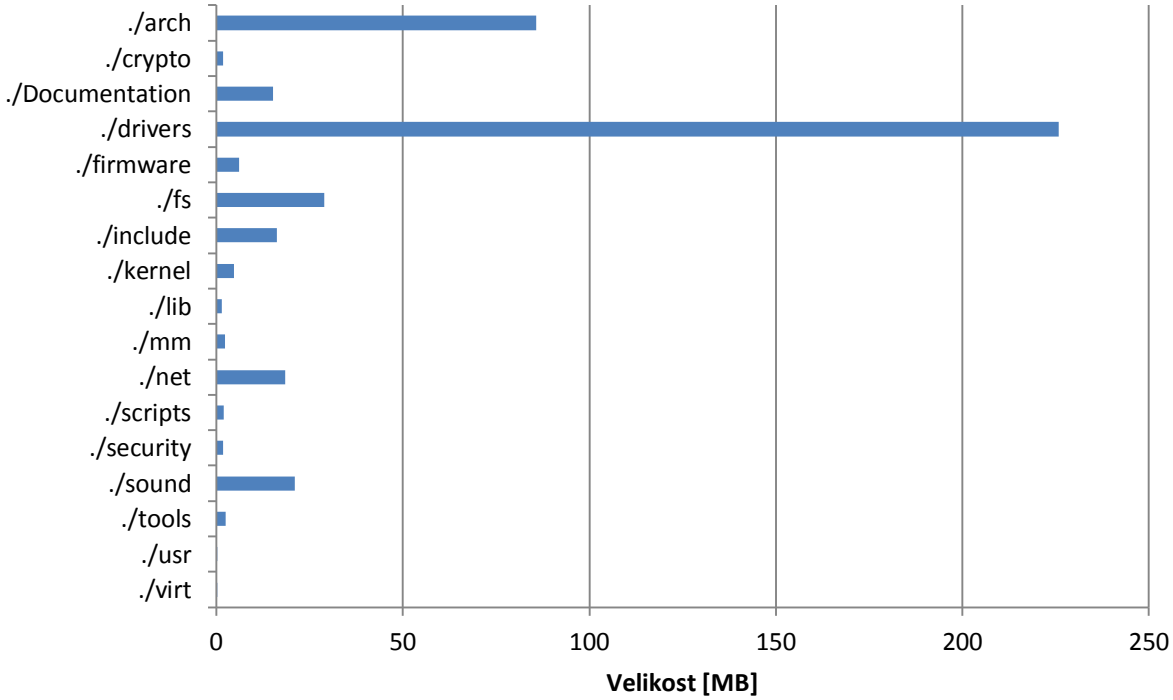
Tab. 1.5 Výkon jader procesoru NIOS II pro konkrétní FPGA obvody (převzato z [3])

2 Úvod do embedded operačního systému Linux

Operační systém Linux lze dnes nalézt nejen na osobních počítačích, serverech a superpočítačích, ale také jako embedded systém na zařízeních, jako jsou mobilní telefony, domácí multimediální centra, kapesní počítače, navigace, lékařská zařízení a další. Rozdíl mezi verzí operačního systému Linux pro osobní počítače a servery a verzí pro embedded zařízení je především v návrhu systému a výběru aplikací. U embedded zařízení je nutno brát ohled na jejich omezené možnosti, jimiž jsou zejména velikost operační paměti, úložného prostoru a výkonu procesoru.

Jádro operačního systému Linux, určeného pro embedded zařízení vychází ve většině případů z běžného jádra, uvolněného panem Linusem Torvaldsem [4]. Toto jádro obsahuje zdrojové kódy k velkému množství ovladačů. Podporuje mnoho síťových protokolů, architektur, souborových systémů, atp. Důsledkem je rapidní nárůst jeho velikosti a nároků na hardware. Velikosti jednotlivých částí jádra jsou patrné z obr. 2.1, který ukazuje velikosti adresářů zdrojového kódu jádra verze 3.3. Současná verze tak může mít velikost až kolem 500 MB. Při pohledu na samostatný základ jádra lze zjistit, že je možné vytvořit funkční operační systém Linux s velikostí něco kolem 1 MB, který je vhodný pro použití právě v embedded zařízeních s omezeným výkonem a pamětí.

Je obecně známo, že kolem operačního systému Linux vystupuje velké množství komunit lidí, kteří často spravují a vyvíjejí vlastní jádro, často přidávají nové možnosti a optimalizují ho pro danou architekturu. Tyto verze jader bývají mnohdy méně stabilní.

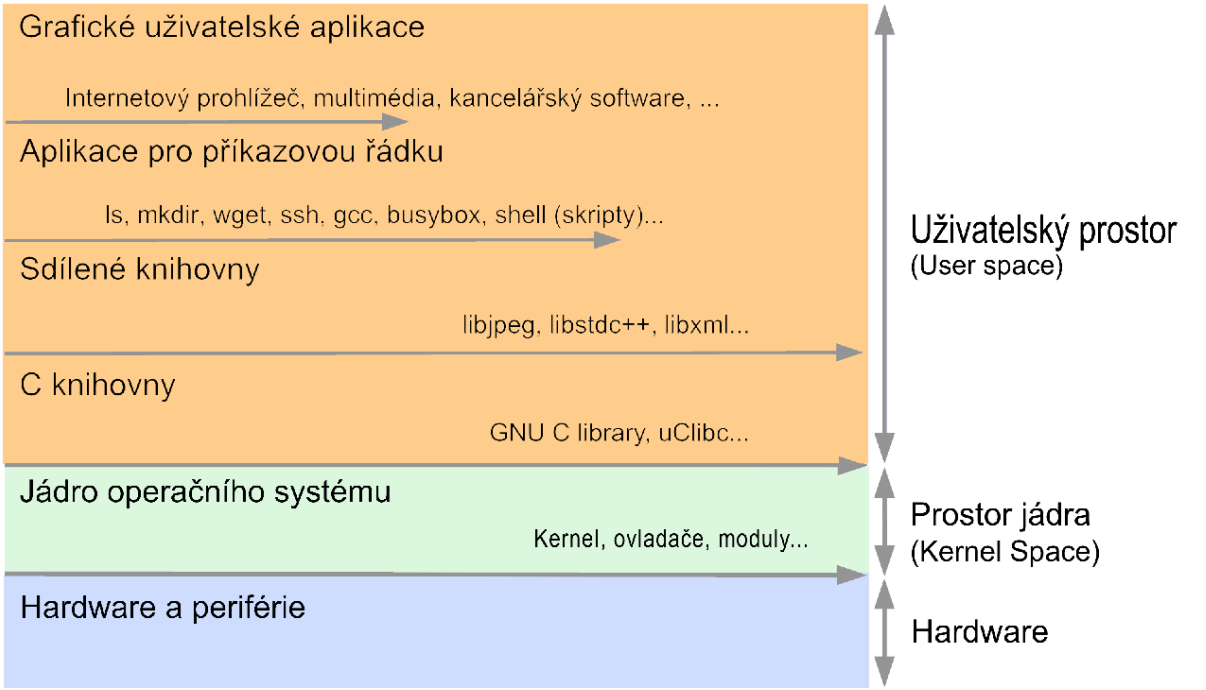


Obr. 2.1 Velikost adresářů se zdrojovým kódem jádra operačního systému Linux (v. 3.3.0)

2.1 Systémová paměť

Systémovou paměť lze v operačním systému Linux rozdělit na dvě odlišné oblasti, *user space* (uživatelský prostor) a *kernel space* (prostor jádra). Prostor jádra je striktně vymezen a z bezpečnostních důvodů umožňuje přístup z uživatelské oblasti pouze za pomocí tzv. systémových volání, která jsou součástí knihoven jazyka C. Prostor jádra je využíván nejen jádrem, ale i většinou ovladačů a modulů. Systémová volání jsou žádosti aktivního procesu o služby prováděné jádrem, jako je vstup/výstup nebo vytváření nového procesu. Aktivní proces je takový proces, který je v dané chvíli zpracováván procesorem. Vstup/výstup může být reprezentován jakýmkoliv programem, operací, nebo zařízením, které komunikuje s procesorem, nebo s některou z periférií (např. USB disk, klávesnice, myš, atd.). V uživatelském prostoru pak běží vše ostatní, jako jsou klasické aplikace, knihovny, atp.

Snahou je, aby v uživatelském prostoru běželo co nejvíce věcí a tím se eliminoval počet chyb v jádře a možnost pádu celého systému. Většina pádů operačních systémů Linux je způsobena hardwarem, nebo chybou ovladačů. Toto rozdělení paměti je důležité zejména u systémů s jednotkou správy paměti (MMU).



Obr. 2.2 Rozdělení paměťového prostoru

2.2 Zavaděč systému

Jádro operačního systému Linux od verze 2.6 používá malý dočasný souborový systém, označovaný jako *initramfs*. Tento souborový systém obsahuje moduly a vlastní jádro. *Initramfs* bývá většinou komprimovaný do formátu gzip. Lze však využít i některý z jiných formátů (např. bzip2, LZMA, LZO). Skripty obsažené v *initramfs* automaticky detekují hardware, načtou požadované moduly a připojí reálný souborový systém. Na závěr jsou spuštěny inicializační aplikace na reálném souborovém systému. *Initramfs* je většinou obsažen přímo v obraze jádra, nebo může být za použití zavaděče nakopírován do pamězi RAM. Celý proces zavádění systému je popsán tabulkou 2.1 [5].

Zavaděč Spuštěn za pomocí hardwaru, z přesně daného umístění ve flash, nebo ROM paměti. Obsahuje podporu pro zařízení, na kterém je uložen obraz jádra (lokální zařízení, síť, vyměnitelná média). Načte obraz jádra do paměti RAM. Spustí obraz jádra.
Jádro (kernel) Archiv jádra se sám rozbalí. Inicializuje se jádro kernelu a staticky zkompilovaných ovladačů. Pokud obraz jádra obsahuje initramfs, dojde k jeho rozbalní. V opačném případě je zkopírován do paměti pomocí zavaděče. Dojde ke spuštění programu v uživatelském prostoru, pokud existuje v umístění /init.
Uživatelský prostor: /init skript může vykonat následující: Spustí příkaz ke konfiguraci zařízení. Připojí souborový systém a přepne na něj. Spustí /sbin/init.
Uživatelský prostor: /sbin/init Spouští příkazy ke konfigurování zbylých zařízení, pokud to již nebylo vykonáno dříve v initiramfs. Spouští systémové služby a uživatelské programy.

Tab. 2.1 Postup zavedení systému

2.3 Nástroje pro zkompilování jádra pro jinou architekturu

Vzhledem k tomu, že ve většině případů se kompiluje jádro operačního systému Linux a ostatní software pro embbeded systémy na jiném zařízení (např. osobní počítač), je potřeba pracovat s takzvanými *toolchain* nástroji. Jedná se o soubor nástrojů, určených pro vývoj softwaru, které jsou určitým způsobem propojeny. Pro embedded systémy je nejzajímavější

toolchain pod označením *cross-native*. Tento bývá sestaven na architektuře x86, ale generuje kód, který lze spustit na jiné cílové architektuře (např. NIOS II, ARM, PowerPC a jiné). Umožňuje tak kompilovat a vyvíjet systém a software pro embedded systémy na osobním počítači [6].

Dále jsou uvedeny komponenty, které může *toolchain* obsahovat:

Binutils

Binutils je první nezbytná součást nástrojů *toolchain* a obsahuje dva velmi důležité nástroje:

- *as, assembler*, který mění kód assembleru (generovaný za pomocí GCC) na binární
- *ld, linker*, který vytváří knihovny a spustitelné soubory
- *Binutils* může dále obsahovat další nástroje pro manipulaci s binárními soubory.

C, C++, Java a další překladače

Druhou nezbytnou součástí nástrojů *toolchain* je překladač jazyka. U operačního systému Linux pro embedded zařízení se dnes používá zejména nástroj GCC (*Gnu Compiler Collection*), který nepodporuje jenom jazyk C, ale i další programovací jazyky, mezi které patří C++, Fortran, Java, Objective-C a Ada. GCC podporuje výstup pro celou řada různých architektur.

Knihovna jazyka C

Knihovna jazyka C (*C Library*) poskytuje tradiční rozhraní pro vývoj uživatelských aplikací, které s jádrem komunikují pomocí systémových volání a poskytují služby vyšším vrstvám. K nejčastěji používaným patří:

- *Glibc*, jedná se o knihovnu jazyka C, kterou používají téměř všechny počítače a servery. Knihovna je vyvíjena v souladu s normami, ale je příliš velká.
- *Embedded glibc (EGlibc)*, jedná se o variantu pro embeddded systémy. Cílem projektu je redukovat velikost a podporovat kompilování a testování softwaru pro jiné architektury, při zachování kompatibility s knihovnou *Glibc*.
- *uClibc*, je alternativní knihovnou jazyka C pro embedded systémy, která má menší velikost a používá se tak zejména u systémů s omezenou velikostí paměti. Obecně se má za to, že knihovna není zatím tak stabilní, jako *Glibc* a neposkytuje tolik možností. Na jejím vývoji se však v poslední době usilovně pracuje.

Volba knihoven jazyka C musí být provedena již při generování *toolchain*. Je na ní totiž vázán kompilátor jazyka a po sestavení sady *toolchain* ji již nelze změnit.

Debugger

Součástí *toolchain* může být ještě navíc ladící software, který umožňuje odladit aplikace běžící na cílové architektuře. Pro embedded systémy se většinou používá *GNU Project Debugger,* neboli GDB.

2.4 Soubor základních nástrojů Busybox

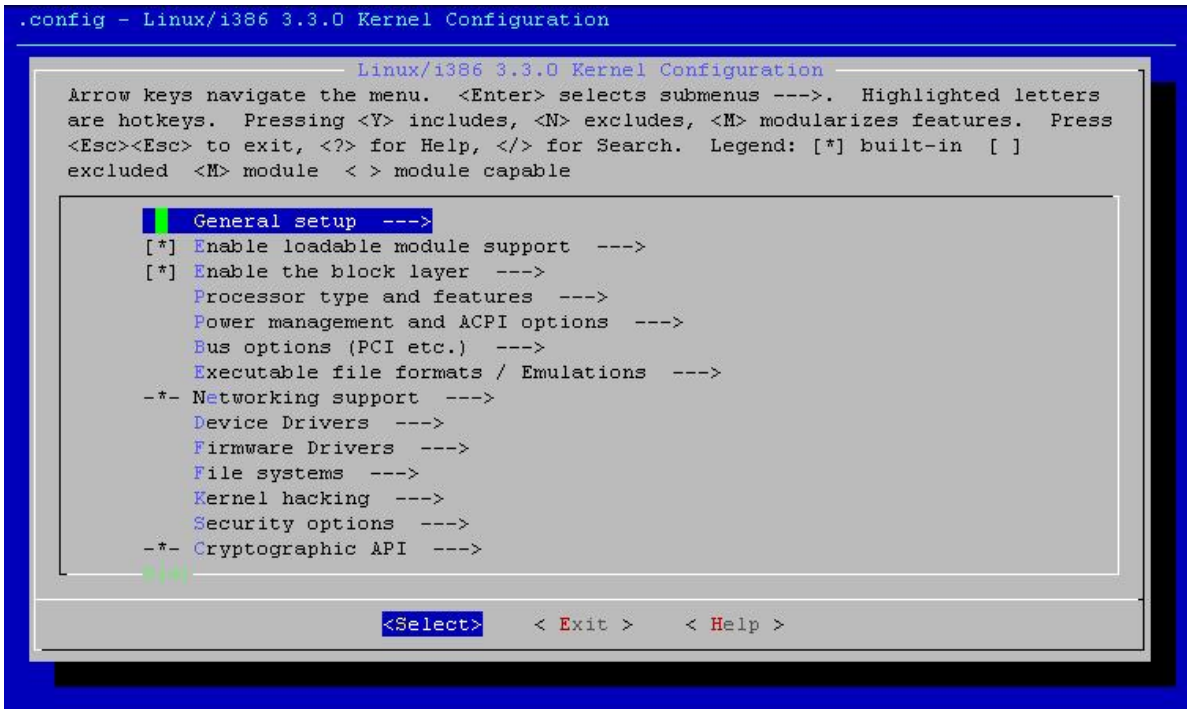
Busybox je svobodný software, který se používá na téměř všech embedded systémech s operačním systémem Linux. Jedná se o základní balík nástrojů pro práci se systémem, které jsou obsaženy v jednom spustielném souboru. Busybox nahrazuje základní příkazy známé z běžných distribucí operačního systému Linux (jako jsou nástroje pro práci s adresáři a soubory, archivační nástroje, základní textový editor, některé nástroje pro komunikaci se sítí, atp.). To vše s ohledem na výslednou velikost tak, aby bylo možné systém spustit na zařízení již se 4 MB paměti. Busybox je velmi modulární a při jeho kompilaci lze jednoduče přidávat, nebo odebírat některé z nástrojů, příkazů a funkcí, které obsahuje. Pro práci s jednotlivými nástroji jsou v systému vytvořeny tzv. symbolické odkazy, které odkazují na spustitelný binární soubor **busybox.** S těmito symbolickými odkazy je následně možno pracovat jako s běžnými příkazy a nástroji, které jsou známé z operačního systému Linux. Nástroje, které Busybox obsahuje, neobsahují všechny možnosti jako nástroje v běžných distribucích. Zachovávají si však svou funkčnost [7].

2.5 Konfigurace jádra operačního systému Linux

Jádro operačního systému Linux obsahuje řadu nejrůznějších ovladačů, podporu souborových systémů, síťových protokolů a dalších služeb. Pro každou verzi jádra tak existuje velké množství voleb, které mohou být použity ke zkompilování daných částí jádra. Proto je zejména u embedded systémů zapotřebí vybrat pouze potřebné volby a tím optimalizovat jádro na co nejmenší velikost a nejlepší výkon.

Konfigurace jádra je uložena v souboru **.config**, v kořenovém adresáři se zdrojovým kódem jádra. Pro pohodlnější konfiguraci se volby nemění ručně v souboru, ale používají se některá grafická rozhraní umožňující snadnou konfiguraci.

Je jím zejména />**make menuconfig**, který umožňuje pohodlné konfigurování jádra v příkazovém řádku. Nástroj MENUCONFIG potřebuje k funkci knihovny *ncurses*.

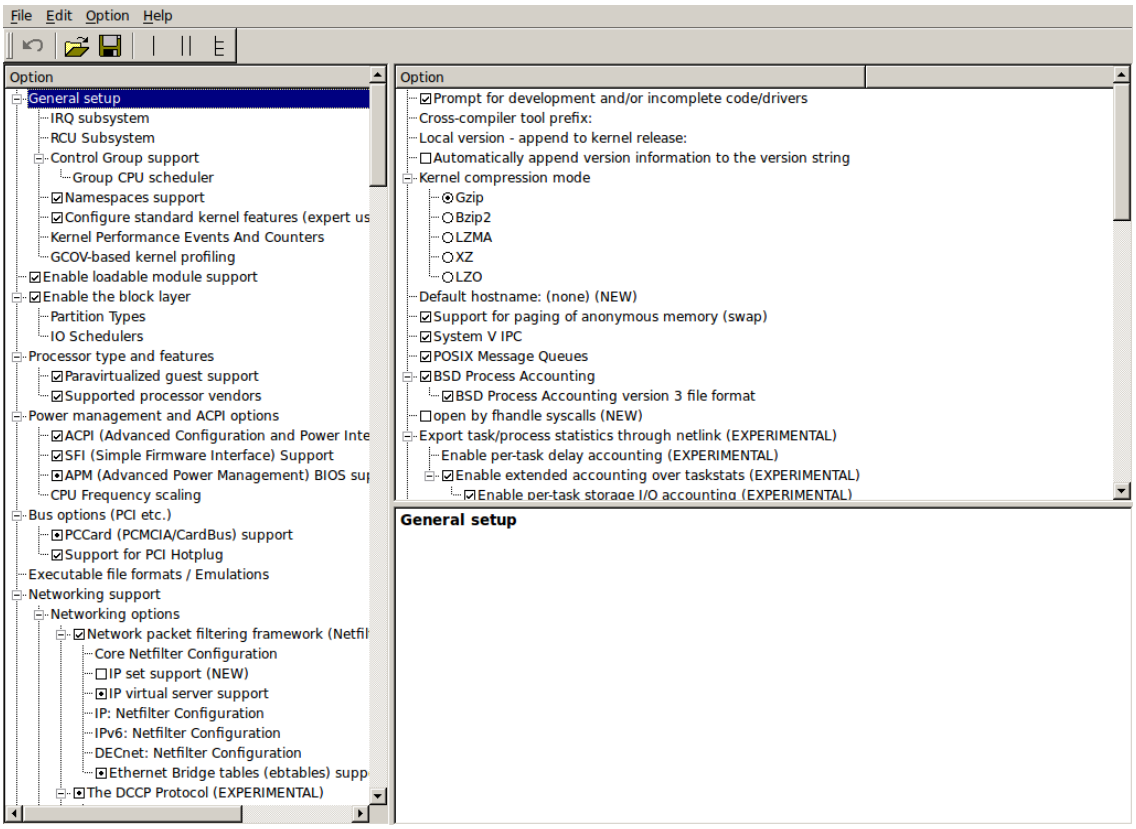


Obr. 2.3 Konfigurační nastroj menuconfig

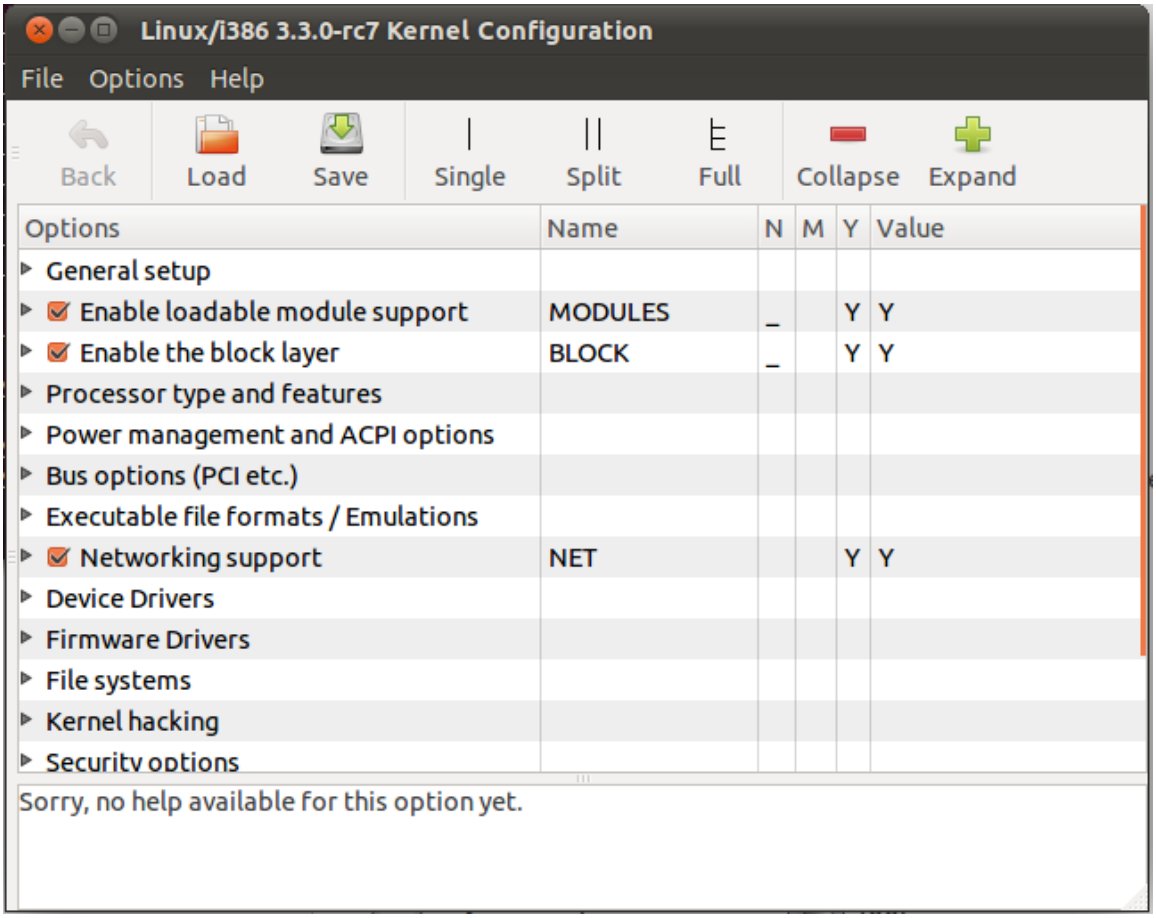
Dalším, často používaným nástrojem pro konfiguraci jádra operačního systému Linux je />**make xconfig**, který spustí konfiguraci v grafickém rozhraní systému. Ke spuštění jsou potřebné knihovny QT a okenní manažer.

Dalším novějším grafickým nástrojem je />**make gconfig**. Tento nástroj využívá knihoven GTK.

Posledním nástrojem, který je nutno zmínit, je />**make oldconfig**, který se používá především v situacích, kdy dochází k přechodu na novou verzi jádra. Měl by být spuštěn tehdy, když dojde k editaci konfiguračního souboru **.config** ručně za použití textového editoru.



Obr. 2.4 Grafický konfigurační nástroj xconfig



Obr. 2.5 Grafický konfigurační nástroj gconfig

2.6 Kompilování jádra operačního systému Linux

Po zkonfigurování jádra se provede zkompilování příkazem />**make**. Na stroji s více jádrovým procesorem lze kompilování urychlit volbou />**make –j 4**. MAKE je nástroj, který byl vytvořen pro automatizaci kompilování zdrojových kódů. Dnes je obsažen ve všech běžných distribucí operačního systému Linux. Příznak -j určuje, že bude docházet k paralelnímu kompilování několika souborů najednou, v tomto případě konkrétně až ke čtyřem. Při tomto současném překládání několika zdrojových kódů dochází k využívání dalších procesorů, popř. jader procesoru. Tímto příkazem dojde k vygenerování obrazu jádra. Název výsledného souboru se může lišit dle cílové architektury a zvolenho komprimačního formátu [8].

3 Tvorba ovladačů a aplikací v prostředí operačního systému Linux

Velkou výhodou při psaní ovladačů a aplikací pro operační systém Linux je skutečnost, že lze volně získat a modifikovat jejich zdrojové kódy, které mohou posloužit nejen jako ukázkové, ale lze použít také celé jejich části při psaní vlastního software. Mezi nejběžnější programovací jazyk v prostředí operačního systému Linux patří C/C++. Výjimkou však nejsou ani jiné jazyky. Pro jednoduché, menší aplikace lze také využít skriptovací jazyk BASH (v prostředí SHELL). Takovýto skript se skládá z příkazů, které lze běžně používat v příkazové řádce systému. BASH však umožňuje také jednoduchou práci s proměnnými, větvení programu, uživatelský vstup a výstup, práci s logickými výrazy a další. Toho se dá využít také v inicializačním skriptu systému. Lze tak spustit služby a aplikace automaticky po startu systému. Dále se dá skriptovací jazyk BASH použít ke psaní CGI skriptů.

3.1 Aplikace v operačním systému Linux

Aplikace v operačním systému Linux jsou reprezentovány dvěma skupinami souborů: spustitelnými soubory a skripty. Operační systém Linux nevyžaduje, aby tyto soubory měly speciální příponu. K tomuto účelu se využívají atributy souborového systému. V operačním systému Linux lze zaměnit spustitelné soubory a skripty, aniž by došlo k ovlivnění uživatelů, nebo jiných programů. Na uživatelské úrovni tak mezi těmito dvěma skupinami souborů není žádný rozdíl.

Po přihlášení do systému uživatel komunikuje s programem SHELL, který za něj spouští programy. Programy, které chce uživatel spustit, vyhledá v zadané sadě adresářů. Tyto adresáře jsou uloženy v proměnné s názvem PATH.

Programy jsou zpravidla umisťovány do adresářů vyhrazených k danému účelu. Adresář </bin> obsahuje binární soubory, které jsou používané při zavádění systému a další důležité systémové nástroje. Adresář </usr/bin> obsahuje většinou programy k obecnému použití, které byly nainstalovány správci systému. V umístění </usr/local/bin> jsou zpravidla instalovány programy pro specifický hostitelský počítač nebo lokální síť. U některých distribubcí pro embedded systémy se lze setkat pouze s adresářem </bin>, který v tomto obsahuje všechny spustitelné soubory.

3.1.1 Sada překladačů GCC

V operačním systému Linux se využívá ve velké míře k překladu programů nástroj *GNU Compiler Collection*, zkráceně GCC. Jedná se o sadu překladačů, která podporuje následující jazyky C, C++, Objective-C, Fortran, Java, Ada, Go. Nástroj také obsahuje knihovny pro podporované jazyky. Tento nástroj se po zavolání automaticky stará o celý proces sestavení programu. Uživatel tak zavolá nástroj s patřičnými parametry, ve kterých lze uvést, jaký jazyk má kompilátor použít, popřípadě o tom nástroj GCC rozhodne sám. Zdrojový kód programu je nejprve zpracován preprocesorem, jehož výstup je dále předán překladači. Výstupem překladače je program v tzv. jazyce symbolických adres, což je v podstatě nízko úrovňový jazyk používající strojový kód pro danou architekturu. V posledním kroku je spuštěn *linker* (sestavovací program), který vytvoří výsledný spustitelný soubor. Nejjednodušší použití nástroje GCC je následující: />**gcc hello.c -o run**. Nástroj GCC v tomto případě použije soubor **hello.c** jako vstupní soubor se zdrojovým kódem. Výsledný spustitelný soubor bude uložen v souboru s názvem **run** [9].

3.1.2 Hlavičkové soubory

Při programování v jazyce C se využívají hlavičkové soubory, které obsahují definice konstant a funkcí. Tyto soubory jsou většinou uloženy v adresáři </usr/include>, popřípadě v jeho podadresářích. Pokud dochází ke kompilaci programu pro jinou architekturu, bude umístění závislé na použitém balíku *toolchain* nástrojů. Většinou se jedná o adresář </include>. Pokud se hlavičkové soubory nacházejí v jiném umístění, lze je překladači zpřístupnit pomocí přepínače -I. Příkaz pro přeložení zdrojového kódu může být například />**gcc -I/usr/include hello.c -o run**.

3.1.3 Soubory knihoven

Na knihovny lze nahlížet, jako na předkompilované funkce napsané tak, aby je bylo možno využívat v různých aplikacích. U operačního systému Linux jsou tyto knihovny nejčastěji uloženy v umístění </usr/lib> a </lib>. Obdobně, jako u hlavičkových souborů, tak také u knihoven lze překladači sdělit, v jakých adresářích má knihovny hledat. V tomto případě k tomu slouží parametr -L.

U systému Linux se rozlišují dva typy knihoven, které se zpravidla liší příponou souboru:

.a reprezentuje tradiční statickou knihovnu

.so a *.sa* reprezentuje sdílenou knihovnu

V případě použití statické knihovny, dojde během procesu kompilování programu k začlenění vlastní knihovny do programu. Použití statických knihoven má tu nevýhodu, že pokud je několik programů obsahujících tutéž knihovnu spuštěno najednou, objeví se v paměti několik kopií funkcí, které taková knihovna obsahuje. Další nevýhodou je, že knihovna je obsažena v těle programu, čímž narůstá i výsledná velikost všech programů.

Tyto nedostatky řeší sdílené knihovny. Pokud program využívá sdílených knihoven, neobsahuje tělo programu vlastní knihovnu, ale pouze odkaz na její umístění. Při spuštění programu pak dochází k řešení funkčních závislostí a volání sdílených knihoven, které jsou v případě potřeby nahrány do paměti. Před využíváním a psaním knihoven pro embedded systémy je potřeba brát v úvahu používaný *toolchain.* Všechny verze překladučů totiž nemusí sdílené knihovny podporovat [10].

3.1.4 Přístup k hardware z uživatelské aplikace

V případě přístupu k hardware z uživatelské roviny je potřeba oddělit zvlášť systémy bez MMU, kdy lze přistupovat k periferiím přímou adresací v paměti a systémy s MMU, kde je situace podstatně složitější.

V případě systému s podporou MMU nelze adresovat fyzickou paměť přímo, ale je potřeba mapovat virtuální paměťový prostor. Přístup k danému zařízení, nebo prostoru lze získat pomocí funkce volání jádra `mmap` a speciálního souboru, který umožňuje přístup do fyzické paměti </dev/map>.

Funkce `mmap` umožňuje mimo jiné manipulovat se soubory. Obsah souboru může vypadat jako pole v paměti. Následné čtení, nebo zápis do segmentu způsobí, že bude systém číst, nebo zapisovat do příslušné části diskového souboru. Funkce vytváří ukazatel na oblast paměti, sdružené s obsahem souboru, k němuž se přistupuje prostřednictvím otevřeného deskriptoru. Syntaxe funkce je následující:

```
void *mmap(void *addr, size_t len, int prot, int flags, int fildes,
off_t off)
```

Parametr `addr` určuje konkrétní adresu v paměti. Pokud je tento parametr nulový, dochází k alokování ukazatele automaticky. To se doporučuje vzhledem k přenositelnosti na jiný systém, jelikož se systémy mohou lišit rozsahem dostupných adres.
Parametr `len`, který musí být několika násobkem délky stránky, kterou vrací `sysconf(_SC_PAGE_SIZE)`, určuje délku oblasti ve slabikách [11].

Příznak `prot`, slouží k nastavení přístupových práv k paměťovému segmentu, jeho volby jsou následující:

- `PROT_READ`, segment je možné číst
- `PROT_WRITE`, do segmentu je umožněn zápis
- `PROT_EXEC`, segment je možné provést
- `PROT_NONE`, nlze přistupovat do segmentu
- Parametr `flags` nastavuje, jak se změny projeví jinde, možnosti jsou:

`MAP_SHARED`, změny budou sdílené, tzn., provedou se i v souboru
`MAP_PRIVATE`, změny nejsou viditelné ostatními procesy, neprovedou se v souboru
`MAP_FIXED`, segment musí být na dané adrese *`addr`*.

- `filedes`, určuje soubor, který je mapován na danou oblast
- parametr `off` určuje offset namapovaného souboru.

Více informací a příklad použití funkce `mmap` lze najít v dokumentaci k jádru.

3.2 Ovladače zařízení

V operačním systému Linux nemusí ovladač nutně řídit přístup k fyzickému hardware ovladače. Existují také ovladače pro tzv. *software devices* (softwarová zařízení). Příkladem může být zařízení </dev/random>, které generuje náhodná čísla. Ovladače zařízení lze dále rozdělit na *character devices* (znaková zařízení) a *block devices* (bloková zařízení). Pro lepší představu lze uvést příklady využití těchto typů ovladačů. V případě, že bude nutné zapisovat data na plotr, nebo tiskárnu, bude zapisování probíhat postupně. To znamená, že budou jednotlivá data zapisována od začátku do konce. Nemělo by smysl používat náhodný přístup do paměti. V tomto případě je možné využít znakové zařízení. Naopak při kopírovaní dat na pevný disk, Flash paměť, či jiné médium není možné pokaždé číst z tohoto média postupně od

začátku. V daném případě bude použití blokového zařízení a náhodný přístup do paměti zcela zásadní.

Přístup ke znakovým zařízením není urychlován vyrovnávací pamětí a čtení nebo zápis probíhají sekvenčně bajt po bajtu. Naproti tomu u blokových zařízení dochází ke čtení v násobcích velikosti jejich bloků a přístup lze realizovat pomocí vyrovnávací paměti.

K většině zařízení lze přistupovat pomocí souborového systému. Výjimku tvoří síťová zařízení, k nimž v operačním systému Linux neexistují speciální souborové záznamy. Speciální soubory ovladačů jsou umístěny v adresáři </dev>. K odlišení těchto souborů ovladačů je použito příznaků `c`, pro soubor ovladače znakové zařízení a `b` pro soubor ovladače zařízení blokového. To je patrné na následujícím, neúplném výpisu z adresáře </dev>.

```
root:/> ls -l /dev
crw-------    1 root     root         5,   1 Jan  1 00:00 console
brw-r-----    1 root     root       179,   0 Apr 25  2012 mmcblk0
```

Číslo, které následuje po vlastníkovi a skupině souboru, je označováno jako hlavní číslo neboli *Major number*. Toto číslo slouží jako jednoznačný identifikátor typu zařízení, následuje číslo vedlejší, které označuje instance zařízení (např. oddíl pevného disku). K vytvoření nového speciálního souboru slouží příkaz />**mknod /dev/test c 112 0**. `/dev/test` udává soubor, který bude vytvořen, `c` značí třídu zařízení (znakové, nebo blokové), `112` reprezentuje hlavní číslo a `0` číslo vedlejší.

3.2.1 Znakové zařízení

Znaková zařízení se registrují v jádře operačního systému Linux a poskytují mu informace, pomocí kterých může jádro spouštět příslušné funkce v okamžiku, kdy aplikace chtějí se zařízením komunikovat. O tuto registraci se stará funkce `register_chardev`, která má následující syntaxi:

```
int register_chrdev(unsigned int major, const chr *name, struct
file_oprations *fops)
```

Funkce vrací zápornou hodnotu při selhání a kladné číslo, nebo nulu při úspěšném vykonání. Parametr `major` udává hlavní číslo. Pokud tento parametr není specifikován, dojde k dynamickému přiřazení tohoto čísla. Dynamické přiřazení je však nepraktické, neboť ke komunikaci se zařízením je nutno toto číslo znát. Je vždy lepší tento parametr stanovit [12].

Čísla 42 a 120-127 jsou rezervována pro lokální zařízení a tudíž se v dalších modulech nevyužívají.

Druhý parametr `name` udává pouze název zařízení, který se následně objeví v souboru </proc/devices>. Poslední parametr je nejdůležitější. Definuje způsob, jakým ovladač komunikuje s okolím, to znamená, které funkce obsluhuje sám a o které se stará jádro. Struktura je definována v hlavičkovém souboru <linux/fs.h>.

Vlastní přístup k zařízení je zprostředkován souborem. Jedná se o speciální soubory, které jsou zcela nezávislé na typu zařízení. Souborových operací existuje celá řada, proto je dále uvedena struktura pouze těch funkcí, které budou použity při psaní ovladačů v dalších kapitolách.

- `write`, tato funkce předává data zařízení. Návratová hodnota je záporná, pokud dojde k chybě. V ostatních případech kladná a udává počet zapsaných bajtů.
- `read`, funkce zapisuje data zpět do uživatelského prostoru. Návratová hodnota je obdobná jako u funkce write.
- `open`, tato funkce je volána, pokud se některá aplikace snaží zařízení otevřít.
- `release`, je volána, pokud dochází k ukončení komunikace se zařízením.

3.3 Ovladače a podpora přerušení

Ne každý hardware je dotazován při řízení datového toku na dostupnost dat a informace o jeho stavu. Dostupnost dat a jiné hardwarové podmínky mohou být signalizovány pomocí přerušení, které může provést příslušnou akci. Jsou definovány dva různé druhy přerušení. Jsou jimi *maskable interrupts* (přerušení, které je možné maskovat) a *nonmaskable interrupts* (přerušení, která nelze maskovat). Pokud je nějaké přerušení maskováno, nedochází k jeho zpracování. *Nonmaskable interrupts* se používají k obsloužení nevratných chyb, které vyžadují okamžitou pozornost systému.

Operační systém Linux zpracovává přerušení následovně: pokud se vyskytne přerušení, je na řadiči zamaskováno, povolí se všechna přerušení na procesoru a vykoná se obslužná rutina přerušení (ISR). ISR dále zavolá příslušný handler přerušení, který obslouží zařízení, které dané přerušení vyvolalo. ISR může být přerušena jinými přerušeními. Ovladač může při

registraci zařízení vynutit, aby při volání ISR daného přerušení došlo k zamaskování přerušení na procesoru. Problémem obsluhy přerušení v operačním systému Linux je absence softwarového maskování přerušení. Pokud ovladač zařízení pracuje se strukturami, které by mohly být přerušením modifikovány, musí být toto přerušení zakázáno. Vzhledem k tomu, že maskování jednoho přerušení na řadiči, bývá pomalé a není možné jej maskovat softwarově, vynucuje ve většině případů operační systém Linux zakázaní všech přerušení [13].

Psaní *Interrupt Service Rutine* (rutin pro obsluhu přerušení), zkráceně ISR, není nikterak složité. Operační systém Linux obsahuje elegantní, nekomplikované rozhraní, které umožňuje registrovat služby přerušení a případně je případně obsloužit tak, jak přicházejí. Obsluha přerušení je do značné míry závislá na architektuře, především na tom, jakým řadičem přerušení je daná platforma vybavena.

Přerušení, která nemají v systému přiřazenou žádnou obsluhu, operační systém Linux pouze oznámí a ignoruje. Seznam registrovaných obsluh přerušení v systému lze najít v souboru </proc/interrupts>. Jeho obsah může vypadat následovně:

```
           CPU0
0:       1875  NIOS2-INTC  timer
1:         39  NIOS2-INTC  altera_jtaguart
4:          2  NIOS2-INTC  isp1362-hcd:usb1
6:          0  NIOS2-INTC  buttons
```

Levý sloupec v tomto výpisu udává číslo přerušení, poté následuje číslo, které udává, kolikrát bylo dané přerušení obslouženo. Zbývající dva údaje představují typ přerušení a jméno zařízení, které bylo zaregistrováno obsluhou přerušení.

Další soubor, který lze využít při vývoji ovladače přerušení je soubor </proc/stat>. V tomto souboru lze najít informaci o počtu všech vyvolaných přerušení, ke kterým od spuštění systému došlo. Řádek obsahující tuto informaci je označen řetězcem `intr` a má následující formát `intr total irq0 irq1 irq2`. `total` znamená součet všech vyvolaných přerušení, `irq0` je počet vyvolaných přerušení od přerušení číslo 0, `irq1` počet vyvolaných přerušení od přerušení číslo 1, atd.

3.3.1 Alokování přerušení

Pro alokování vlastního přerušení se v operačním systému využívá následující funkce

```
int request_irq(unsigned int irq, void (*handler)(int, void *, struct
   pt_request), unsigned long irqflags, const char *devname, void *dev_id)
```

Po úspěšném provedení vrací funkce `request_irq` hodnotu 0 a selhání je označeno zápornou návratovou hodnotou. Tato záporná hodnota může být –EBUSY, pokud došlo k požadavku na sdílenou obsluhu přerušení a pokud již parametr `irqflags` (viz dále) neodpovídá zavedené obsluze přerušení. Další návratovou hodnotou může být –EINVAL, je-li přerušení mimo rozsah [10]. Parametry funkce jsou:

`irq` Konkrétní IRQ, které chcete obsloužit.

`handler (int irq, void *dev_id, struct pt_regs *reg)`
Tato funkce se zavolá, když dojde k přerušení. Funkce se nazývá *handler* přerušení.

`irqflags` Parametr, který určuje chování přerušení.

`devname` Název, pod kterým bude přerušení registrováno (uveden v /proc/interrupts)

`dev_id` Tento parametr se předává výše uvedené funkci handler. Pomocí tohoto parametru lze obsluze přerušení předat informace.

Pokud dojde k přerušení, obdrží registrovaný *handler* tři argumenty. Argument `irq` se využívá pouze tehdy, když jedna stejná *handler* funkce řídí větší množství přerušení. V opačném případě přesně víme, které přerušení funkci vyvolalo. Argument `regs` není příliš často využíván. Obsahuje obraz registrů CPU před tím, než došlo k vyvolání přerušení. Posledním argumentem je `dev_id`, který byl již uveden výše.

Odregistrování funkce *handler* se provádí funkcí `free_irq`, jejíž parametry jsou obdobné jako u předchozí funkce. Není proto nutné jejich další vysvětlení.

```
void free_irq(unsigned int irq, void *dev_id)
```

3.3.2 Funkce pro obsluhu vyvolaného přerušení (*handler*)

Funkce *handler*, která je volána po vyvolání přerušení, musí vhodným způsobem obsloužit zařízení, které toho přerušení vyvolalo. Většinou se jedná o data, která je nutno ze zařízení přenést, popřípadě došlo k nějaké stavové změně, o které chce dané zařízení informovat.

Pokud dojde k výskytu přerušení, dochází k zastavení normálního provádění úloh v systému, to znamená, že jádro pozastaví svoji činnost a je zavolán příslušný handler přerušení.
Handler přerušení by neměl trvat příliš dlouho, protože by mohlo dojít k zmeškání dalšího přerušení, které může zařízení vyvolat. Následující kód definuje ukázku handleru přerušení [14]:

```
1: void irqreturn_t(int irq, void *dev_id, struct pt_regs *regs)
2: {    int status;
3:
4:      printk("received interrupt: %d\n",irq);
5:
6:      //zakazani preruseni
7:      outl(0, IRQ_MASK_PORT)
8:
9:      // precteni stavu zarizeni
10:     status = inl(STATUS_PORT);
11:
12:     /* V pripade potreby prenos dat ze zarizeni a
13:     vykonani potrebnych operaci */
14:
15:     //povoleni preruseni
16:     outl(-1, IRQ_MASK_PORT)
17:
18:     //navratova hodnota, prerusni bylo obslouzeno
19:     return IRQ_HANDLED;  }
```

4 Možnosti nasazení OS Linux na procesor NIOS II

V poslední době se na internetu objevilo několik nových embedded distribucí operačního systému Linux pro softwarové procesory. Konkrétně pro procesor NIOS II je možno využít projekty od společností Wind River, Timesys, SLS a dále také distribuci vyvíjenou komunitou lidí, která se zabývá procesory NIOS II. Poslední zmíněná distribuce je dostupná zdarma pod označením *NIOS II community edition*. Tato distribuce je založena na distribuci uClinux.

4.1 Distribuce uClinux

Projekt *Micro-Controller Linux* (uClinux) je potřeba uvést zejména proto, že z něho vychází mnoho dalších distribucí operačního systému Linux pro embedded systémy. Jeho cílem bylo spustit operační systém Linux na zařízeních bez podpory *Memory Management Unit* (jednotky správy paměti) a vytvoření kompletní distribuce, včetně knihoven a aplikací. Projekt uClinux založili v roce 1998 Jeff Dione a Kenneth Albanowski. První pokusy probíhaly na procesoru Motorla 68328 Dragonball, který byl použit v prvním zařízení typu *PocketPC* (kapesního počítače). Procesor měl taktovací frekvenci 16 MHz. Použité jádro bylo verze 2.0.33. Po těchto úspěšných pokusech se kolem distribuce utvořila komunita lidí, která přidala podporu pro další procesory (Motrorola ColdFire, procesory ARM a AXIS ETRAX) [15]. Dnes již distribuce uClinux podporuje mnoho různých architektur, jejichž výčet je následující (Vývoj projektu stále probíhá a neustále jsou přidávány další nové architektury. Z tohoto důvodu již seznam nemusí být aktuální.[1]):

- Altera NIOS
- Analog Devices Blackfin[2]
- ARM (silicon from Atmel, NetSilicon, Aplio, TI, Samsung, Conexant, a další)[3]
- AXIS ETRAX
- Hitachi H8/300
- Hitachi Super SH2
- Intel i960
- MIPS (Brecis, …)

[1] Aktuální seznam podprovaných architektur na: URL: http://www.uclinux.org/ports/
[2] Aktuální seznam podporovaných Blaskfin procesorů na: URL: http://docs.blackfin.uclinux.org/doku.php
[3] Aktuální seznam podporovaných ARM procesorů na: URL: http://www.linux-arm.org/LinuxKernel/WebHome

- Motorola 68k family (68x302, 68306, 68x328, 68332, 68360)
- Motorola ColdFire (5206, 5206e, 5249, 5272, 5282, 5307, 5407)
- Motorola MCORE
- NEC v850 family
- OpenCORES OpenRISC (FPGA)
- Sparc LEON
- Xilinx Microblaze (FPGA processor)

Při vývoji distribuce bylo nutné udělat nejvíce změn v samotném jádru operačního systému Linux. Celé jádro bylo převzato a doplněno o podporu pro systémy bez MMU. *Micro-Controller Linux* má podporu pouze pro 32-bitové a 64-bitové mikroprocesory.

V současné době existují dvě stabilní verze této distribuce, které jsou založeny na jádrech verze 2.0.39 a 2.4.22. Vývojová verze distribuce uClinux obsahuje jádro verze 3.x.x. Všechna tato jádra plně podporují *multi-tasking* (současný běh několika procesů) s běžným modelem jejich řízení. *Micro-Controller Linux* pracuje s běžnými typy souborových systémů a síťových protokolů, které jsou známé z operačního systému Linux pro běžné počítače. *Micro-Controller Linux* dokonce obsahuje i originální ovladače zařízení a podporuje jejich dynamické načítání.

Klasická koncepce rozdělení paměti na *user space* (uživatelský prostor) a *kernel space* (prostor jádra) je zachována přesto, že jádro neobsahuje podporu pro správu paměti. Pokud hardware nepodporuje řízení přístupu paměti nebo ukazatelů, musí softwarovou simulaci těchto funkcí pro systémy bez MMU zajišťovat jádro operačního systému. Zásadním rozdílem mezi standardní distribucí operačního systému Linux a distribucí uClinux je zejména to, že systém nepodporuje žádnou formu virtuální paměti. To znamená, že všechny aplikace mohou využívat pouze paměť RAM. Distribuce uClinux nepodporuje použití žádného odkládacího prostoru ve formě oddílu, nebo souboru na externím úložném zařízení. Vlastní systém alokování paměti je zachován z běžné verze jádra. Jedinou změnou je, že alokátor nechává prázdné velké bloky paměti, tak aby byly připraveny při požadavku na jejich alokaci. Při požadavku na alokování paměti musí alokátor najít jeden „souvislý kus“ paměti, který musí být dostatečně velký na to, aby vyhověl danému požadavku. Tento systém je nutný, protože není možné dohromady virtuálně mapovat stránky paměti a tím vytvořit velký region vhodný pro alokování.

Distribuce uClinux obsahuje místo standardní Glibc knihovny jazyka C, knihovnu uClibc, která má několika násobně menší velikost. uClinux dále obsahuje celou řadu známých knihoven, mezi které patří openssl, zlib, libjpeg, libpng a mnohé další.
Aplikace jsou v distribuci uClinux načítány a spouštěny stejným způsobem jako v kterékoliv jiné běžné distribuci. Aplikace tedy skládá ze stejných základních částí. Jedná se o *code portion* (část kódu), někdy také označovanou jako textový segment, *initialized data section* (inicializační datovou sekci), často označovanou jako datový segment, *uninitialized data section* (neinicializační datovou sekci), neboli bss a zásobník [16].

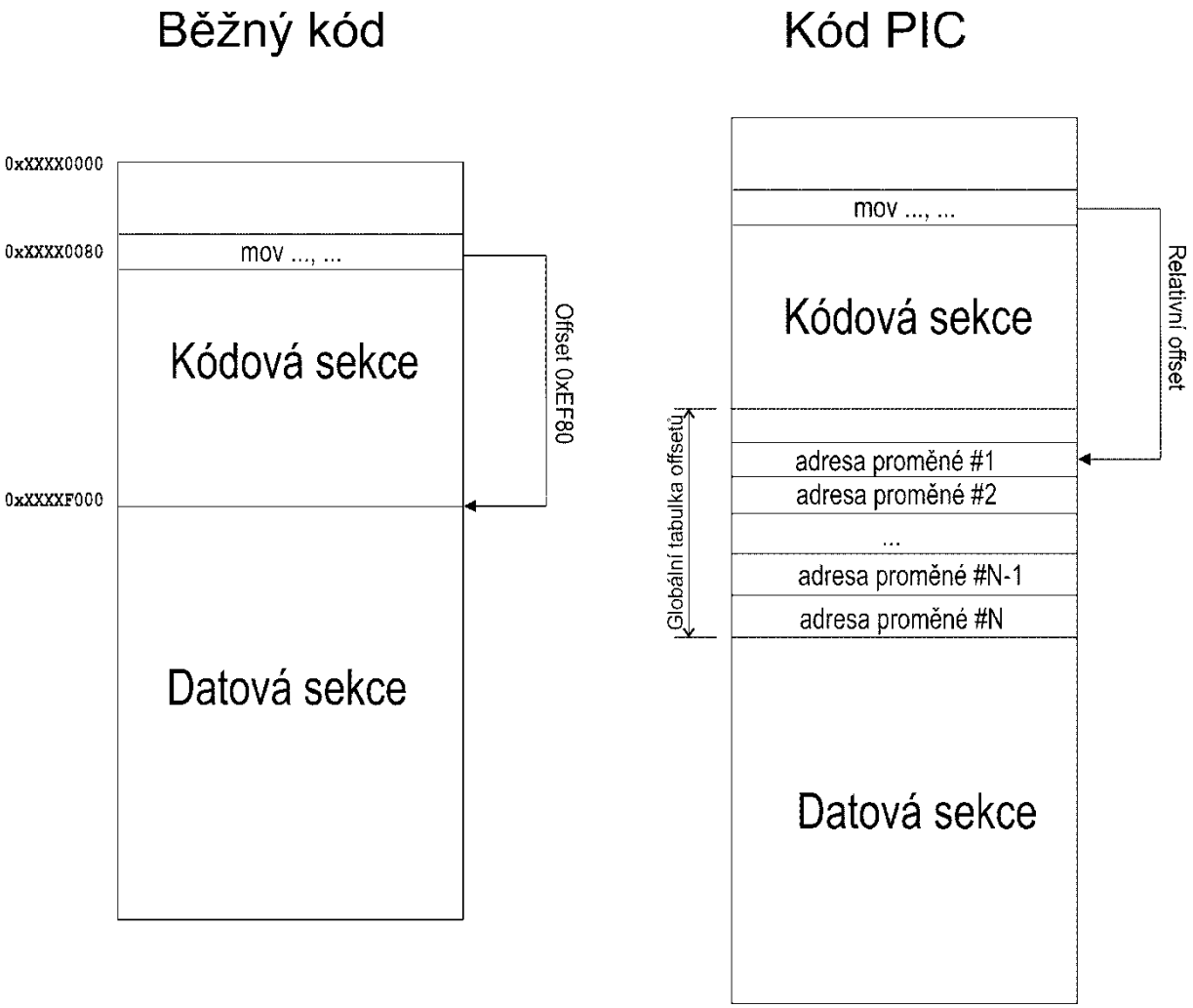
Distribuci uClinux přináší podporou možnosti ponechat část kódu aplikace v původní paměti (např. Flash, nebo ROM) a spustit instrukce z tohoto paměťového prostoru. To se nazývá *execute in place* (spuštění na místě), Zkratkou se tento termín označuje jako XIP. Tímto způsobem lze uspořit podstatnou část paměti. Aby bylo možné dosáhnout této úspory, musí být celý kód programu uložen v jednom souvislém bloku paměti. Tuto vlastnost musí zajistit souborový systém.

Aplikace obsažené v distribuci uClinux používají nový binární formát, který se nazývá *flat file*. Důvody pro zavedení tohoto formátu jsou dva. Jedním z nich je potřeba jednoduše a rychle zavádět a spouštět procesy pro aplikace. Druhým důvodem je potřeba generovat velmi malé binární soubory, z důvodu výsledné velikosti distribuce.

U systémů, využívajících virtuální paměť, je pro každou aplikaci vyhrazen prostor v této virtuální paměti. Adresy v rámci kódu a dat tak mohou být v tomto prostoru paměti přesně definovány. V distribuci uClinux nejsou žádné pevně stanovené adresy, proto může být aplikace načtena a spuštěna z kterékoliv oblasti paměti RAM. V případě spuštění aplikace za použití systému XIP (z nějakého místa v paměti Flash, ROM, nebo jiného úložného zařízení), nelze předem určit adresu paměti, na které se bude vyskytovat daný kód. Tento problém řeší uClinux dvěma způsoby.

Prvním z nich je způsob přemístění. Přemístitelné položky jsou uloženy v binárním formátu *flat file*. Pokud dochází k načítání programu, je v jádru spuštěn tzv. *flat loader*. Ten přemístí aplikaci do volného bloku paměti a upraví kód a data aplikace pomocí adresy této vyhrazené paměti (tzn., že adresy alokované paměti pro danou aplikaci boudou shodné s adresou přemístění). Tuto metodu nelze využít u systému XIP.

Druhou metodou je tzv. metoda kódu nezávislého na umístění (PIC). U této metody musí kompilátor sestavit kód, který je nezávislý na umístění v paměti. Jedná se o kód, který neobsahuje žádný absolutní odkaz na adresu v paměti. Ke spuštění aplikace metodou XIP nestačí mít takový kód. Dále je potřebná ještě pozičně nezávislá datová část programu. Toho je ve většině případů dosaženo globální tabulkou offsetů, ve které je pro každou adresu definován offset. Globální tabulka offsetů je součástí datové sekce. Strukturu kódu PIC ukazuje obr. 4.1. Je zde porovnán kód PIC s kódem běžné aplikace. Přístup do paměti je řízen zvláštním registrem. PIC kódy bývají zpravidla o něco pomalejší, protože nemají přímý přístup do paměti. Mají však tu výhodu, že je lze použít u systému XIP [17].



Obr. 4.1 Porovnání kódu PIC s běžným kódem programu

Dá se říci, že metoda PIC je u distribucí pro embedded systémy více populární. Není však podporována na všech architekturách a vyžaduje překladač, který umí takový kód a data vytvářet. Metodu přemístění v paměti je snadnější implementovat, proto je tato metoda využívána při vývoji a implementaci operačního systému Linux na nové architektury.

Metody *flat file* a PIC se vzájemně nevylučují a mohou být společně použity v jednom systému. Jádro z hlavičky souboru zjistí, zda lze aplikaci spustit systémem XIP, nebo ne.

Jak již bylo zmíněno, výhodou formátu *flat file* je jeho malá velikost. Záhlaví souboru je velké 40 B a uvnitř souboru nejsou použity žádné mezery a přebytečná data.

Vzhledem k tomu, že distribuce používá originální jádro operačního systému Linux, které je doplněno o záplaty a funkce, lze v systému s výhodou využít běžné aplikace. V základní distribuci uClinux tak lze najít velké množství programů a knihoven. Následující tabulka uvádí některé z nich.

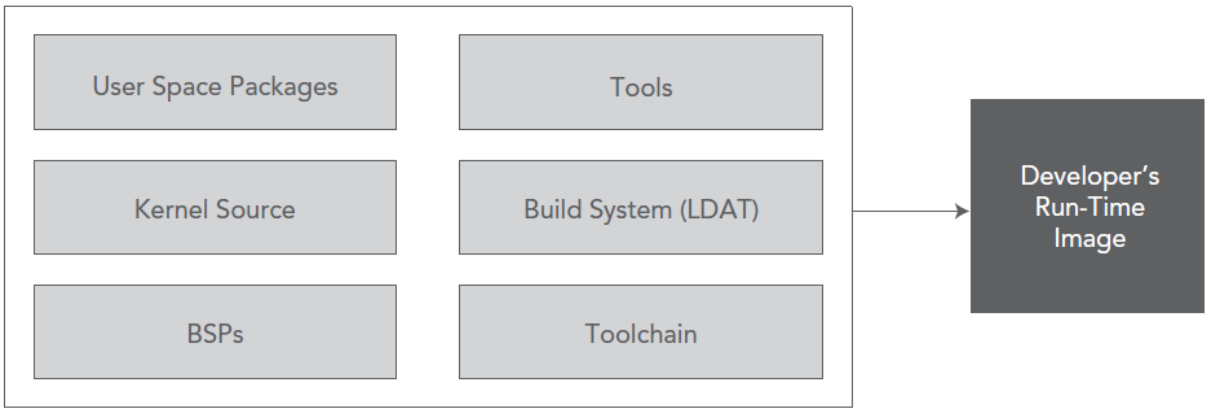
Nástroje/Utility	busybox, mhash shell, python, vi
Nástroj pro práci se sítí	net-tools, ping, iptables, ftp, dhcpd, traceroute, ntp, wget, iproute2, tcpdump
Servery	inetd, pptpd, boa, telnetd, tftpd, nfsd, samba, squid, sshd, snmpd, dnsmasq
Nástroje pro práci se souborovým systémem	mount (podporuje NFS), smbmount, fdisk, e2fsprogs, reiserfs tools
Ostatní	mp3play, microwindows, mtd-utils

Tab. 4.1 Seznam několika aplikací, které obsahuje distribuce uClinux

4.2 Distribuce Wind River Linux

Společnost Wind River úzce spolupracuje se společností Altera. Nabízí se tak zajímavá, profesionální distribuce, která je přizpůsobená procesoru NIOS II. Tato distribuce je testována vývojáři a nabízí spolehlivost a stabilitu. Obsahuje všechny komponenty, které vývojáři aplikací požadují. Společnost ke svému produktu nabízí technickou podporu a profesionální servis.

Aktuální verze Wind River 4 je založena na jádře 2.6.34 a podporuje vývojové desky s čipem CYCLONE III [18].



Obr. 4.2 Hlavní komponenty Wind River Linuxu (převzato z [18])

4.2.1 Sestavení distribuce Wind River Linux, nástroj LDAT

Pro sestavení systému používá Wind River Linux nástroj zvaný *Linux Distribution Assembly Tool (*LDAT*)*. Nástroj je licencován pod GNU licencí. LDAT běžně řeší proces sestavení, který je uveden na obrázku 4.2.

Proces sestavení distribuce začíná vytvořením projektu za použití nástroje LDAT. Uživatel specifikuje základní informace, které jsou potřebné ke konfiguraci jádra, jako je architektura, typ jádra, je možné vybrat konfiguraci pro konkrétní vývojovou desku, provést konfiguraci uživatelského prostoru, atd. LDAT na základě této konfigurace automaticky vybere vhodnou verzi jádra a balík nástrojů *toolchain* pro kompilaci distribuce na cílovou architekturu. Dále je možno za pomoci nástroje LDAT přidat do projektu veškeré vlastní komponenty a ovladače. Přidání vlastních komponent a ovladačů je v distribuci Wind River Linux označováno jako tzv. *Custom Layer* (Vlastní vrstva).

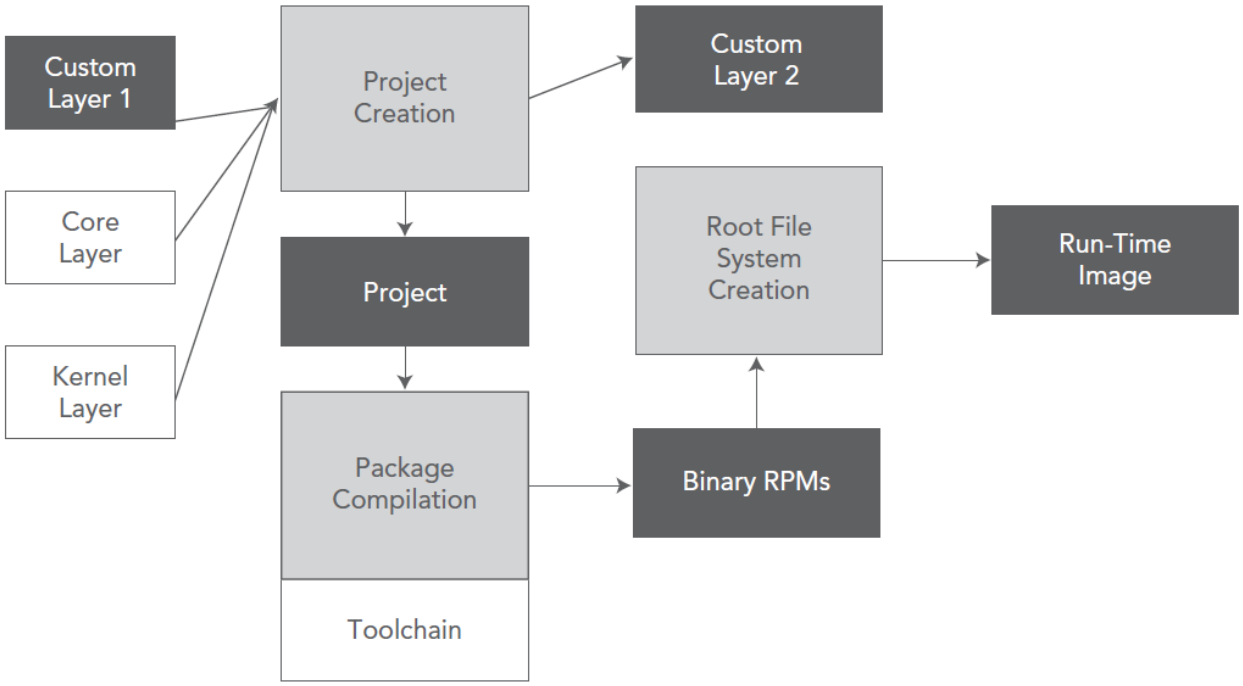
Jednou z výhod nástroje LDAT je, že po sestavení systému dokáže vlastní vrstvy vyexportovat a posléze použít v jiných projektech. LDAT poté provede všechny potřebné lokální změny konfigurací, nástrojů a balíčků v novém projektu.

Jak již bylo řečeno, používá nástroj LDAT hierarchický systém vrstev, které mohou obsahovat vše od běžných konfiguračních souborů, balíků aplikací, až po složité věci, jako je nahrazení jádra systému, rozšíření nástrojů toolchain, atd. Základní vrstvy, které distribuce Wind River Linux obsahuje, jsou následující.

- *Core Layer*, obsahuje veškeré aplikace pro uživatelský prostor
- *Kernel Layer*, obsahuje zdrojové kódy jádra
- *Toolchain layer*, obsahuje nástroje *toolchain*, pro sestavení systému na jiné architektuře

Další, uživatelsky definované vrstvy mohou pro urychlení vývoje obsahovat místní změněné soubory a předkompilovaný software.

Konečný kořenový systém souborů je sestaven z jednotlivých balíčků programů a komponent. LDAP využívá balíčkovací systém *RPM package manager*, který je všeobecně známý z distribucí RedHat, CentOS, Arch Linux a dalších. Systém umožňuje automatické řešení závislostí mezi jednotlivými balíčky knihoven a aplikací. Systém je možno sestavit znovu od základu nebo pouze aktualizovat přidáním, odebráním nebo aktualizováním určitých RPM balíčků v rámci ušetření času při vývoji.



Obr. 4.3 Sestavení Wind River distribuce (převzato z [18]).

4.2.1.1 Základní vlastnosti distribuce Wind River Linux

Základem současné verze Wind River Linux je jádro verze 2.6.34. Wind River přidává do tohoto jádra mnoho nových funkcí a opravuje některé chyby. Takto upravné jádro je poté dlouhodobě testováno za účelem dosažení co možná nejstabilnějšího operačního systému.

Toolchain, použitý v distribuci Wind River Linux, je modifikací jiného balíku *toolchain* a to sice Sourcery G++. Verze, která byla dostupná v den psaní této práce, obsahuje GCC verze 4.4, Binutils verze 2.19 a GDB verze 7.0.

Výchozí používanou knihovnou jazyka C je eGlibc, na jejímž vývoji se společnost Wind River přímo podílí. Je však dostupná i verze s uClibc.

Dále distribuce nabízí podporu pro nejrůznější hardware, knihovny a software pro zabezpečení systému. Nástroje pro analýzu paměti, výkonu, času zavádění systému a další testy. Distribuce také obsahuje více než 550 nejrůznějších uživatelských aplikací, mezi nimiž nechybí webový server, podpora pro databáze, sdílení souborů a další .

4.2.2 Distribuce od společnosti Timesys (LinuxLink)

Společnost Timesys nabízí distribuci operačního systému Linux pro embedded systémy pod označením LinuxLink. Tato distribuce však přichází zatím pouze s podporou čipů Cyclone III a běžných vývojových desek s touto řadou čipů. Jsou dostupné verze s jádrem 2.6.30 a 2.6.21. Společnost v rámci své obchodní politiky nabízí tři následující verze distribuce [19].

LinuxLink Free je verze, kterí umožňuje vývojářům rychle implementovat operační systém Linux do referenčního či vývojového kitu, aniž by potřebovali jakékoliv vývojové prostředí. Tato verze má všechny aplikace a knihovny překompilovány a dochází tak k sestavení distribuce podle zvolené platformy a řady procesoru. Toto sestavení se provádí na vzdáleném cloud serveru. Uživatel je následně upozorněn e-mailem s odkazem, na kterém si může svou distribuci stáhnout. Tato verze je vhodná zejména pro začínající uživatele. Verze *Linux Link Free* je poskytována zdarma.

LinuxLink TimeStorm ID zachovává všechny výhody předchozí verze. Dále přidává nástroje pro snadnější a rychlý vývoj aplikací, s možností ladění, optimalizování výkonu a provádění testů. Vývojové prostředí TimeStorm IDE obsahuje mimo jiné nástroje *Cross toolchain Plugin* pro správu verzí nástrojů *toolchain, Target Management Plugin* pro snadné stahování, spouštění a ladění systému na cílové architektuře, nástroj GDB a další. Cena roční license je u této verze $1795.

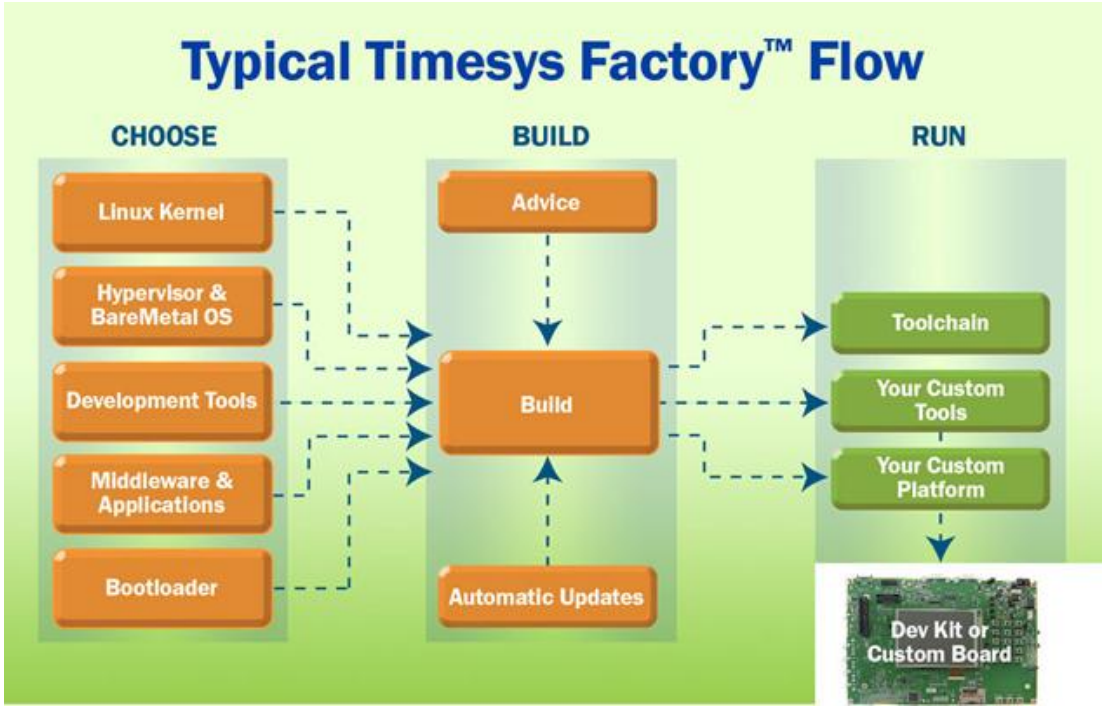
LinuxLink Pro je nejvyšší řada produktu, kterou firma Timesys poskytuje. Je určena především pro průmyslové použití. Přichází s celou řadou již předkompilovaných a testovaných aplikací a kompletní sadou vývojových aplikací. Jedná se o aplikace, které by mohli vývojáři systémů potřebovat. Společnost na svých stránkách slibuje celkový čas třiceti

minut od získání distribuce, až po její spuštění v embedded systému. Společnost dále uvádí následující hlavní výhody této distribuce:

- Systém je sestaven za použití běžných nástrojů operačního systému Linux, jako je MAKE.
- Zajišťuje, že jedna změna nevyžaduje další rozsáhlé změny kódu.
- Poskytuje ucelený soubor jader a balíků, které jsou již zkompilované a udržováním aktuálních informací o vývoji systému, umožňuje v kterékoliv etapě vývoje zpětně sestavit distribuci podle některé z dřívějších konfigurací.

Cena roční license u této verze činní $5495.

Společnost Timesys vyvinula pro sestavení embedded distribuce vlastní nástroj pod označením *Timesys Factory*. Proces sestavení distribuce je naznačen na následujícím obrázku 4.3. Tento nástroj umožňuje sestavit distribuci ze zdrojového kódu, přičemž se snaží o maximální flexibilitu a možnost vlastních změn. Umožňuje jednoduchým způsobem začleňovat vlastní aplikace. Analyzuje konfiguraci systému a závislosti aplikací za účelem identifikace možných problémů, včetně návrhů jejich řešení. Automaticky se stará o aktualizování systému. Ke konfiguraci využívá standartní nástroje (KCONFIG, MAKE, atd.). Podporuje mnoho různých architektur (ARM, Power, MIPS, NIOS II, x86, atd.). Dále umožňuje vybrat jednu z možných C knihoven (uClibc, Glibc, eGlibc, Klibc) a jeden z možných balíčkovacích systémů (RPM, DEB, atd.).



Obr. 4.4 Sestavení distribuce Timesys LinuxLink (převzato z [20]).

4.2.3 Distribuce od společnosti System Level Solutions pro některé vývojové desky společnosti Altera

Společnost System Level Solutions je dalším partnerem společnosti Altera. System Level Solutions nabízí na svých stránkách hotové linuxové distribuce pro následující vývojové desky s procesorem NIOS II.: Altera EDK Cyclone III, Altera Startix IV GX FPGA Devlopment kit, NEEK a CoreCommander. Systém Level Solutions nabízí tyto distribuce ke stažení zcela zdarma. Na stánkách společnosti je možné najít jak sestavený a zkompilovaný obraz distribuce se souborem .sof pro konfigurování procesoru, tak i zdrojové kódy distribuce [21].

Distribuce pro vývojovou desku CoreCommander je postavena na distribuci uClinux a obsahuje jádro verze 2.6.28 a GCC verze 3.4.6. Distribuce dále obsahuje ovladače a podporu pro následující periférie, které jsou obsažené na desce: síťový adaptér, řadič pro SD karty, řadič USB 2.0, podporu pro Framebuffer na LCD displeji a ovladač tlačítek. Balík, který je možno stáhnout ze stránek společnosti, obsahuje referenční design v programu Quartus verze 8.1, zdrojové kódy a hotový obraz distribuce. Jako ukázku obsahuje ještě tyto tři demonstrační aplikace *chat* server, prohlížeč JPEG obrázků a aplikaci na čtení, zápis a ověření dat na USB zařízení.

Další distribucí vycházející z projektu uClinux, kterou lze najít na stránkách společnosti, je pro vývojový kit NEEK. Verze použitého jádra v této distribuci je 2.6.27. Opět lze získat jak hotový obraz distribuce, který lze do desky okamžitě nahrát a spustit a zdrojové kódy. V distribuci nechybí podpora pro periférie obsažené na desce (síťový adaptér, podpora pro PS2 klávesnici, dotykový displej, řadič VGA a SD karet). Nechybí ani ovladač zvukového zařízení. Jako ukázku obsahuje archív s distribucí také výsledný obraz operačního systému Linux, který funguje jako přehrávač hudebních souborů. Druhý z příkladů ukazuje možnosti běhu několika služeb zároveň, neboli *multitasking*. Tento příklad obsahuje webový server BOA, *chat* server a aplikaci, která zaznamenává odeslaná a přijatá data.

Další podporovanou vývojovou deskou je *Embedded Systems Development Kit* (vývojový kit pro embedded systémy), neboli Cyclone III Board. V této distribuci je použito jádro verze 2.6.30 a GCC verze 4.1.2. Distribuce obsahuje podporu pro všechny důležité periférie, které deska obsahuje.

Poslední deskou, pro kterou lze ze stránek společnosti získat kompletní distribuci je vývojový kit Stratix IV GX FPGA Development Kit. Je zde použito jádro verze 2.6.34. I v tomto případě je možné získat nejen zkompilovaný obraz distribuce a referenční design, ale i zdrojové kódy.

Všechny výše uvedené distribuce obsahují BusyBox, jako základní balík aplikací a příkazů.

4.2.4 Upravená distribuce uClinux pro procesor NIOS II

Jedná se o distribuci uClinux, která je podporovaná komunitou lidí, která se zabývá procesory NIOS II. Distribuce navíc obsahuje některé ovladače, které lze najít na vývojových deskách se softwarovým procesorem NIOS II. V současné době je možné tuto distribuci spustit jak s podporu MMU, tak i bez podpory MMU. Podpora jednotky pro správu paměti je hlavním rozdílem oproti originální distribuci uClinux. V dalších kapitolách bude tato distribuce popsána podrobněji. Aktuální informace o této distribuci lze nalézt na fóru společnosti Altera [22] a diskusní skupině *nios2-dev* [23]. Některé starší informace a návody poskytuje server altera wiki [24].

5 Použitý hardware

Jako hardware pro demonstraci spuštění a použití embedded distribuce operačního systému Linux na FPGA procesoru NIOS II byl v této práci použit vývojový kit DE2-70 od společnosti Terasic.

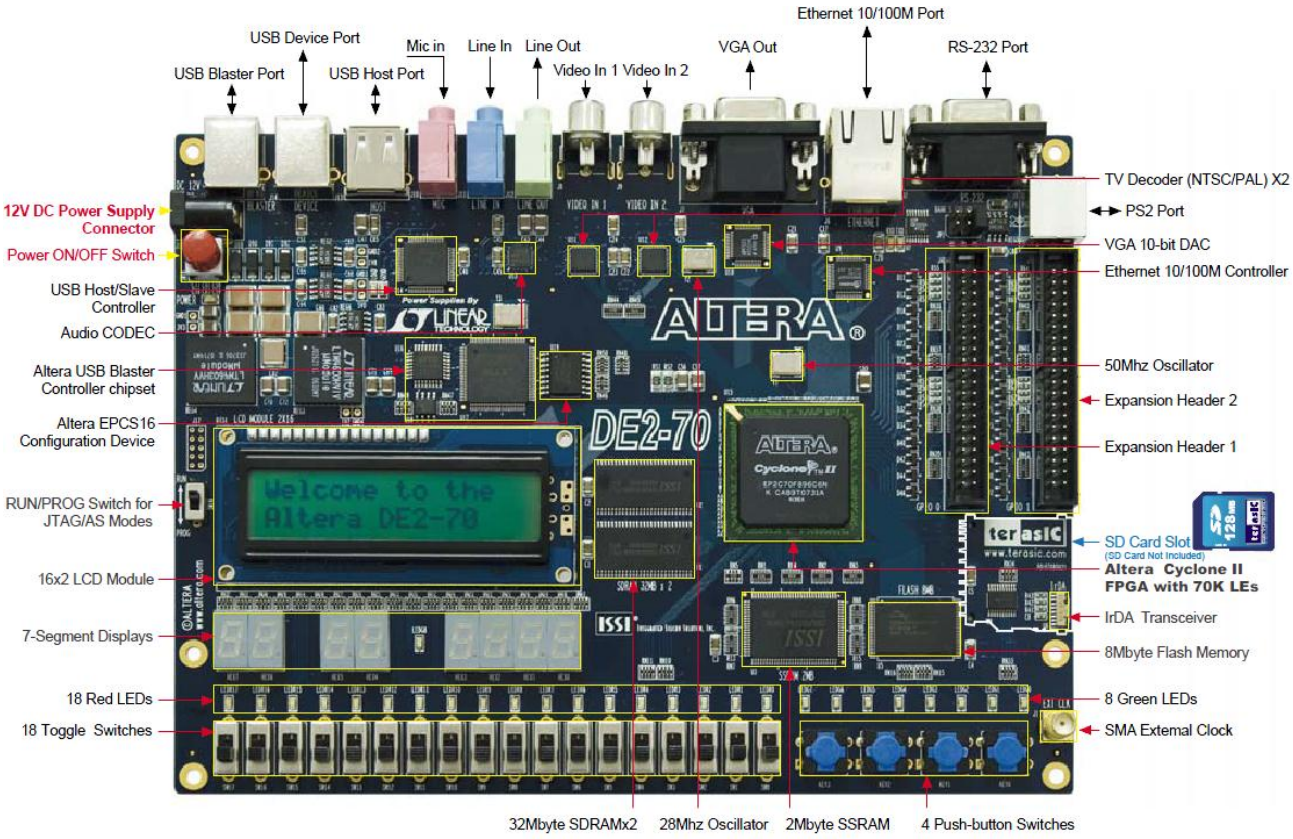
5.1 Popis vývojového kitu DE2-70

Vývojový kit DE2-70 obsahuje výkonný čip Cyclone II FPGA. Na jednotlivé piny tohoto čipu jsou připojeny všechny důležité periférie vývojového kitu, což uživateli umožňuje konfigurovat propojení mezi nimi. Pro použití softwarového procesoru NIOS II je třeba vývojový software od firmy Altera. Pro vygenerování systému se softwarovým procesorem slouží nástroj SOPC Builder nebo Qsys. Nástroje jsou součástí software Quartus II. Pro tuto práci byla použita verze Quartus II 11.1 Build 173. DE2-70 obsahuje následující hardware:

- FPGA procesor Altera Cyclone II 2C70
- Sériové konfigurovatelné zařízení ALTERA EPCS16
- USB Blaster
- 2 MB SSRAM
- 2x 32 MB SDRAM
- 8 MB paměť Flash
- Patici pro paměťové karty typu SD
- 4 tlačítka
- 18 přepínačů
- 18 červených a 9 zelených LED diod
- Oscilátory 50 MHz a 28,63 MHz
- 24bit audio převodník
- 10bit vysoko rychlostní DAC převodník pro VGA výstup
- 2x televizní dekodér podporující normy NTSC, PAL a SECAM
- 10/100 Mbit síťový adaptér
- Řadič USB
- Rozhraní RS-232
- PS2 konektor pro myš, nebo klávesnici
- Infračervený přijímač a vysílač

- SMA konektor
- 2x 40 pinový konektor pro připojení dalšího hardware s diodovou ochranou

Obrázek 5.1 ukazuje vývojový kit a všechny jeho periférie. Podrobný popis všech komponent a vlastností je uveden v uživatelském manuálu [25].



Obr. 5.1 Vývojový kit Altera DE2-70 (převzato z [25])

6 Spuštění operačního systému Linux na vývojové desce DE2-70, distribuce NIOS II Community Edition

Jak již bylo uvedeno, tato práce se bude dále zabývat zejména distribucí, která je spravována komunitou vývojářů na procesorech NIOS II. Výhodou této distribuce je, že je poskytována zcela zdarma. Další výhodou je, že dříve její vývoj a testování probíhaly také na vývojové desce DE2, která je předchůdcem desky DE2-70. V distribuci je tak možné najít ovladače pro některé periférie, které vývojová deska obsahuje.

6.1 Získání a rozbalení nejnovější dostupné distribuce

Poslední verze distribuce NIOS II Comunity linux, jež byla v době psaní této práce dostupná, byla verze 20100621. Obsah tohoto archivu je poměrně zastaralý, lze ho však aktualizovat pomocí nástroje GIT. K získání poslední verze je nutné postupovat následovně [26]:

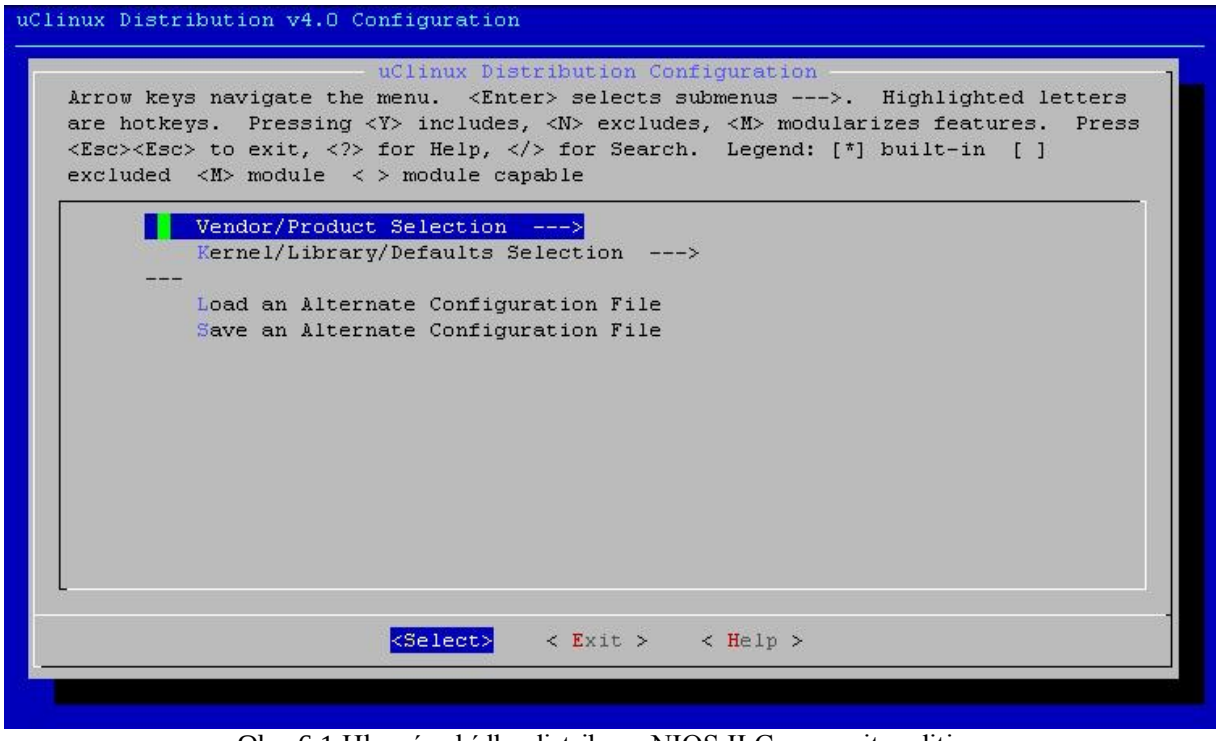
1. Na stroji s operačním systémem Linux je nutné stáhnout soubor **nios2-linux-20100621.tar**, který lze nalézt na DVD, které je přílohou této práce. Soubor je taktéž možné nalézt na FTP serveru altera (http://www.niosftp.com/pub/linux/). Tento archiv je potřeba rozbalit **/>tar -cvf nios2-linux-20100621.tar.**
2. Je nutné rozbalit GIT repositáře />**. /checkout** v adresáři <nios2-linux>.
3. Je třeba stáhnout poslední dostupné jádro z projektu NIOS II Comunity linux. Nejprve je třeba změnit GIT repositář. Ke dni psaní této práce byl název posledního pracovního repositáře „nios2“. Aktuální informace lze nalézt na internetových stánkách projektu altera wiki [8]. Změnu repositáře kernelu lze provést zadáním příkazu />**git fetch origin && git checkout nios2 && git branch –d nios2mmu** v adresáři <nios2-linux/linux-2.6> .
4. Dále je nutné zaktualizovat zbytek distribuce. />**cd uClinux-dist && git pull**.

6.2 Konfigurace distribuce

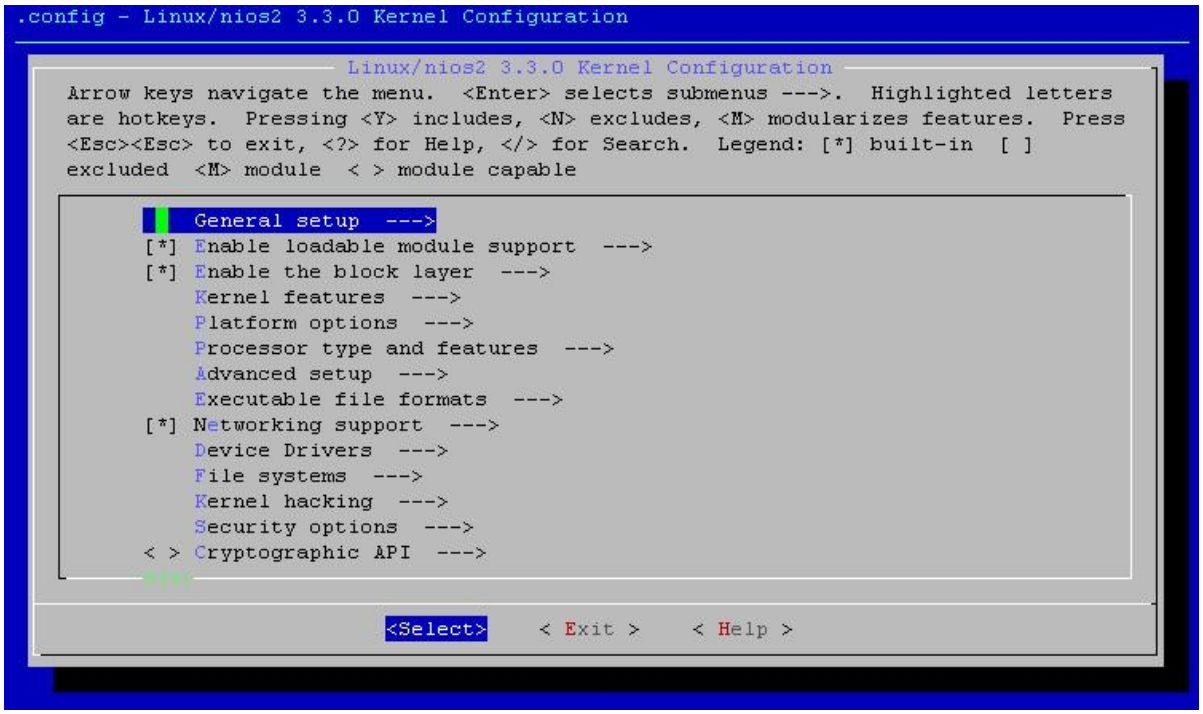
Stejně tak, jako originální distribuce uClinux, obsahuje i tato modifikovaná distribuce dvě hlavní konfigurační nabídky. Jedna z nich je využita pro konfiguraci jádra. Jedná se o modifikaci originálního konfiguračního menu jádra operačního systému Linux (viz kapitola 3.5), která je doplněná o volby specifické pro požadovanou architekturu a speciální ovladače. Naopak volby, které jsou specifické pro běžné počítače, jsou z tohoto menu vynechány.

Druhá nabídka obsahuje seznam veškerého software, který distribuce obsahuje. Je v něm možno zvolit, které programy, nástroje a knihovny budou součástí výsledného sestaveného obrazu distribuce. Na popředí těchto dvou konfiguračních menu je taktéž menu pro konfiguraci výrobce a typu procesoru. Ke konfiguraci distribuce lze tedy využít jeden z konfiguračních nástrojů uvedených v kapitole 3.5. Pro ukázky bude v této práci dále využíván nástroj MENUCONFIG.

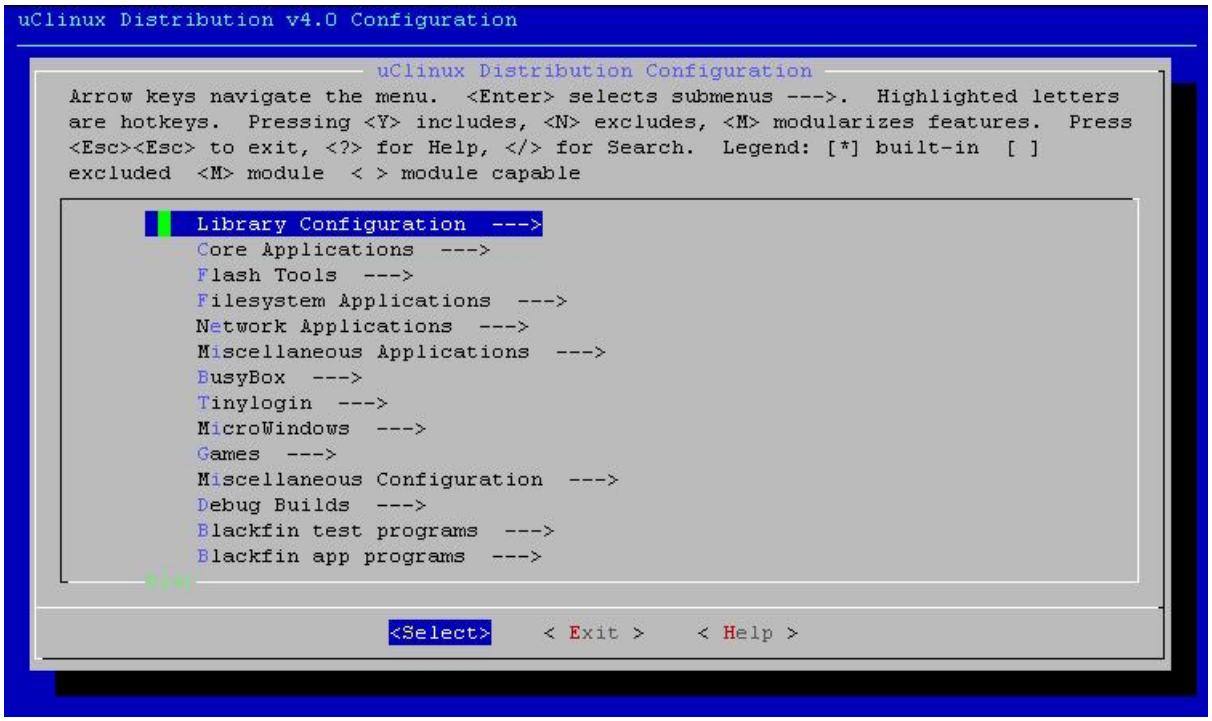
Konfigurační menu distribuce lze spustit příkazem />**make menuconfig** v adresáři <nios2-linux/uClinux-dist/>. Po spuštění tohoto příkazu se zobrazí hlavní nabídka (viz. Obr. 6.1). Záložka *Vendor/Product Selection* obsahuje volby pro výběr procesoru. V případě procesoru NIOS II je potřeba nastavit *Vendor (Altera)* a následně *Altera products.* Zde je možno vybrat z voleb *nios2* nebo *nios2mmu*, pro systém s podporou MMU nebo bez podpory MMU. Záložka *Kernel/Library/Defaults selection* obsahuje dvě důležité volby. *Customize Kernel Settings* pro změnu voleb konfigurace jádra a volbu *Customize Application/Library Settings* pro výběr knihoven, aplikací a nástrojů, které budou zabudovány do výsledného obrazu distribuce. Další volby, které lze v této záložce najít jsou *Default all settings* pro obnovení výchozí konfigurace a *Update default vendor Settings* pro přepsání této výchozí konfigurace distribuce.



Obr. 6.1 Hlavní nabídka distribuce NIOS II Community edition



Obr. 6.2 Nabídka pro konfiguraci voleb jádra



Obr. 6.3 Nabídka pro výběr aplikací a knihoven

Pokud je potřeba do distribuce zahrnout další, vlastní ovladač nebo aplikaci, lze je přidat do jedné z těchto dvou nabídek. Toho lze docílit následujícím způsobem [27]:

1. Nejprve je nutné vhodný adresář, podle toho, do které záložky bude umístěna nová volba. Např. v případě potřeby přidat novou volbu do záložky Drivers->Misc v konfigurační nabídce jádra, bude pracovním adresářem <nios2-linux/linux-

2.6/drivers/misc/>.

2. Dále je třeb přepnout do tohoto adresáře />**cd ./nios2-linux/linux-2.6/drivers/misc/**
3. Je nutno otevřít soubor **Kconfig** a nakonec souboru před `endif #MISC_DEVICES` je potřeb přidat následující řádky:

```
config NOVY_OVLADAC
      tristate "Novy Ovladac"
      help
            Ovladac noveho zarizeni pro DE2_70.
```

   Položka `config` značí jméno pro novou konfigurační volbu, `tristate` představuje název, který bude uveden v konfigurační nabídce. Parametr `help`, který je nepoviný obsahuje text, který může obsahovat nápovědu k této volbě. Tuto nápovědu je možno zobrazit v jednom z konfiguračních nástrojů.

4. Dále je nutno otevřít soubor **Makefile** a přidat následující řádek:

```
obj-$(CONFIG_NOVY_OVLADAC) += novy_ovladac.o
```

   Za řetězcem `CONFIG_` následuje název uvedený v direktivě `config` v souboru **Kconfig,** a `novy_ovladac.o je` soubor ovladače.

V konfiguraci jádra by se měla následně objevit tato nová položka:

záložka *Device Drivers → Misc devices*

[*] *Novy Ovladac (NEW)*

6.3 Použití vhodného balíku nástrojů toolchain

Výběr a použití balíku nástrojů *toolchain* pro architekturu NIOS II závisí na tom, zda je požadavkem sestavení systému s podporou nebo bez podpory MMU. Tyto nástroje lze spustit pouze v prostředí operačního systému Linux. U systému s podporou MMU je situace o něco jednodušší, protože archiv distribuce již tyto nástroje obsahuje a lze je najít v následujícím umístění <nios2-linux/toolchain-mmu/x86-linux2>.

Pro systém bez podpory MMU je potřeba získat starší verzi těchto nástrojů. Tyto nástroje jsou k dispozici buď na přiloženém DVD, nebo online (http://www.niosftp.com/pub/gnutools/nios2gcc-20080203.tar.bz2). Tento archiv stačí pouze rozbalit do libovolného adresáře.

V případě obou verzí je potřeba přidat cestu k nástrojům balíku *toolchain* mezi adresáře, ve kterých operační systém Linux hledá spustitelné soubory. K tomu slouží speciální proměnná PATH. Nastavení se provede následujícím příkazem:

/>export PATH=$PATH:/cesta/k/toolchain/bin

V příkazu je potřeba nahradit `/cesta/k/toolchain` absolutní cestou k balíku nástrojů *toolchain.* Tento příkaz je potřeba opakovat po každém přihlášení do systému. Pro automatické přidání cesty do proměné PATH je potřeba tento příkaz přidat do souboru <~/.bash_profile> [28].

6.4 Informace o hardwaru - struktura device tree

Device tree je datová struktura, která je určená k popisu hardwaru systému. Je schopna popsat počet a typ procesorů, adresy a velikosti RAM pamětí, sběrnice, připojení periferních zařízení, přerušení, řadiče a další [29].

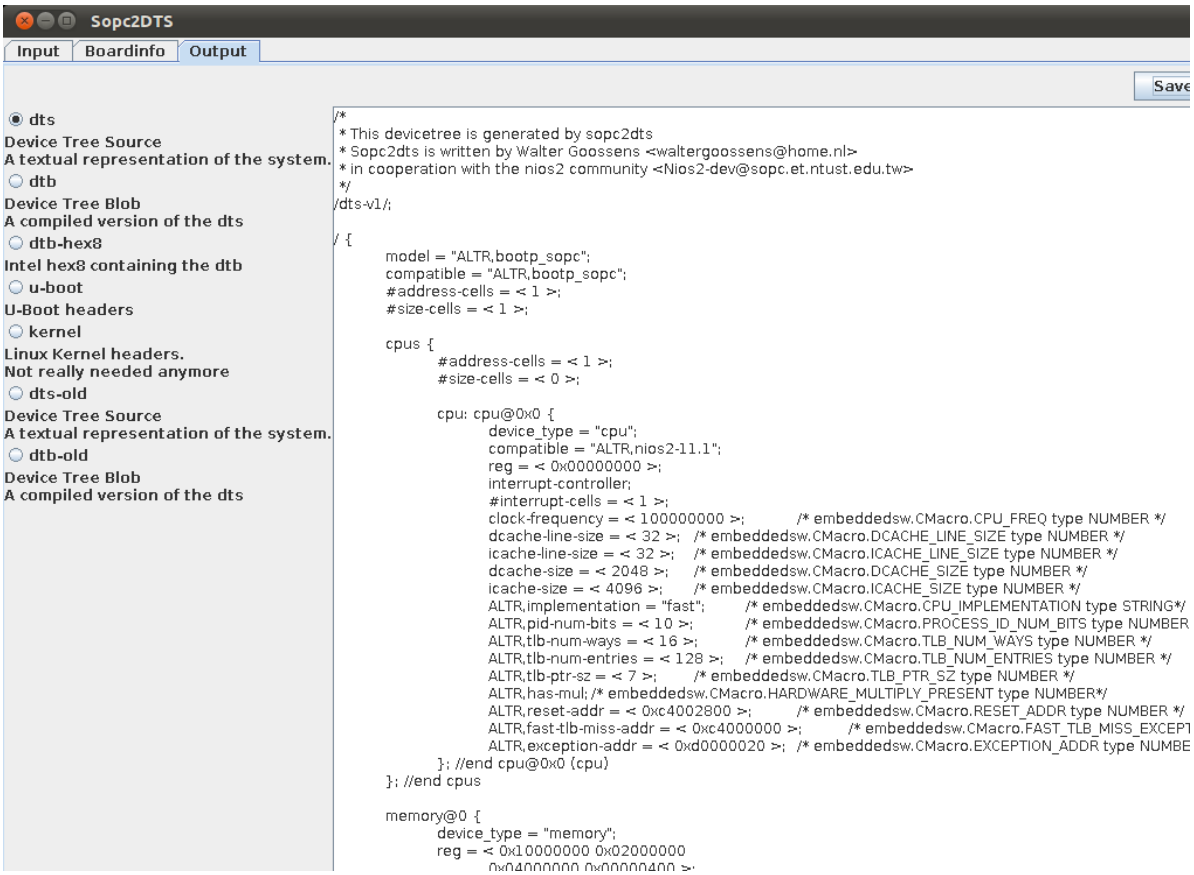
Soubor s device tree strukturou má příponu „.dts“ a lze ho vygenerovat ze souboru „sopcinfo“, který je výstupem buď programu SOPC builder, nebo Qsys. Tento soubor obsahuje informace o komponentách a jejich připojení. K vygenerování souboru s device tree strukturou slouží program SOPC2DTS, který napsal v prostředí Java Walter Goossens [24]. Tento program lze nalézt na přiloženém DVD, nebo stáhnout z GIT repositáře:

/>git clone git://sopc.et.ntust.edu.tw/git/tools.git

Program lze zkompilovat následujícím příkazem za předpokladu, že jsou na stroji nainstalovány nástroje Java development kit:

/>cd tools/sopc2dts && make.

Program SOPC2DTS lze ovládat buď z příkazové řádky, nebo pomocí grafického rozhraní. Použití v prostředí příkazové řádky je následující: />**java -jar sopc2dts.jar -t dts -i system.sopcinfo -o system.dts**. Parametr `system.sopcinfo` je soubor vygenerovaný programem SOPC builder, nebo Qsys a `system.dts` je výstup, neboli výsledná struktura device tree. Grafické rozhraní programu lze spustit zadáním příkazu />**java -jar sopc2dts.jar –gui**



Obr. 6.4 Grafické rozhraní nástroje Sopc2DTS

Po vygenerování souboru s příponou „.dts“, je možné tento soubor ještě upravit v běžném textovém editoru. Při sestavování systému je poté tento zdrojový soubor kompilován a vzniká nový soubor s příponou „.dtb“. Device tree kompilátor se označuje zkratkou DTC. V posledním kroku je takto zkompilovaný soubor přímo implementován do jádra operačního systému nebo ho jádru předá zavaděč systému.

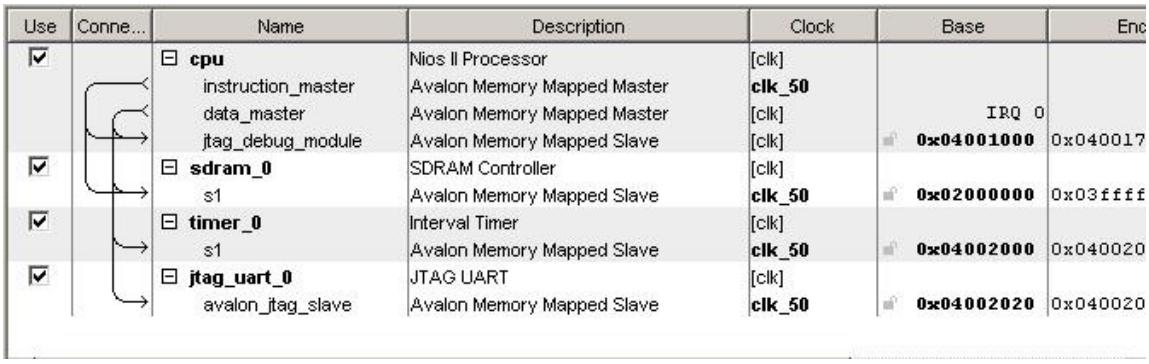
Device tree strukturu lze v poslední verzi distribuce NIOS II Community Edition použít pro systém s podporou MMU i systém bez podpory MMU.

6.5 Spuštění systému bez podpory MMU

Přesto, že se doporučuje používat systém s podporou MMU, což je také výchozí volba při sestavování poslední verze distribuce NIOS II Linux Community Edition. Přináší systém bez podpory MMU stále několik výhod. Jsou jimi zejména větší rychlost systému a snadný přístup k zařízení a do paměti z uživatelských aplikací. Nevýhodou je, že uživatel tohoto systému, může nevhodným zápisem do paměti snadno způsobit pád systému. Proto je potřeba pečlivě zvážit, kde bude takový systém aplikován a kdo k němu bude mít přístup.

6.5.1 Návrh systému

Jak již bylo uvedeno v předchozí kapitole o device tree struktuře, lze k návrhu systému využít nástroje SOPC Builder, nebo Qsys. Pro první seznámení s distribucí NIOS II Community Edition, bude stačit velice jednoduchý návrh systému. Příklad v programu SOPC Builder pro vývojovou desku DE2-70 je uveden na následujícím obrázku. Projekt lze také nalézt na přiloženém DVD v adresáři </examples/boot_nommu_min>. Je důležité nastavit u časovače vlastnost *full-featured* a časový limit 10ms.



Use	Conne...	Name	Description	Clock	Base	End
☑		**cpu**	Nios II Processor	[clk]		
		instruction_master	Avalon Memory Mapped Master	**clk_50**		
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	**0x04001000**	0x040017
☑		**sdram_0**	SDRAM Controller	[clk]		
		s1	Avalon Memory Mapped Slave	**clk_50**	**0x02000000**	0x03ffff
☑		**timer_0**	Interval Timer	[clk]		
		s1	Avalon Memory Mapped Slave	**clk_50**	**0x04002000**	0x040020
☑		**jtag_uart_0**	JTAG UART	[clk]		
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_50**	**0x04002020**	0x040020

Obr. 6.5 Základní návrh systému bez MMU v programu SOPC builder

Pro spuštění distribuce NIOS II Comunity Edition bez podpory MMU lze dále postupovat následovně:

1. Po stisknutí tlačítka *Generate* v nástroji SOPC Builder, nebo Qsys dojde k vygenerování několika souborů, mimo jiné také souboru s příponou „.sopcinfo“.
2. Pro takto vytvořený soubor „.sopcinfo“ je třeba vygenerovat soubor s device tree strukturou (viz kapitola 6.4).
3. V adresáři <./nios2-linx/uClinux-dist> je třeba spustit jeden z nástrojů pro konfiguraci jádra (viz kapitola 3.4) a vybrat následující volby:

   Záložka Vendor/Product Selection

   Vendor (Altera)

   Altera Products (nios2nommu).

   Záložka Kernel/Library/Defaults Selection

   [*] Default all settings

   [*] Customize Kernel Settings
4. Dále je nutné uložit změny a opustit okno s konfigurací. Konfigurační skript se bude dotazovat na některá další nastavení jádra. Pro začátek lze u všech potvrdit výchozí nastavení.

5. Poté se zobrazí nabídka pro konfiguraci jádra. Zde je potřeba nastavit zejména následující parametry:

   Záložka Platform options

   (0x02000000) *Memory base address* (Nutno nastavit na base address SDRAM, uvedené v návrhu systému.)

   [*] Compile and link device tree into kernel image

   (/cesta/k/dts/souboru) *Device tree source file* (nutno zadat úplnou cestu k souboru)

   Takové nastavení postačuje k prvnímu spuštění operačního systému Linux na procesoru NIOS II, lze však rovnou přidat i ovladače pro periférie, či jiné volby.
6. Je nutné nainstalovat a nastavit cestu k vhodnému balíku nástrojů *toolchain* (viz kapitola 6.3).
7. Je nutné zkompilovat distribuci příkazem />**make**.

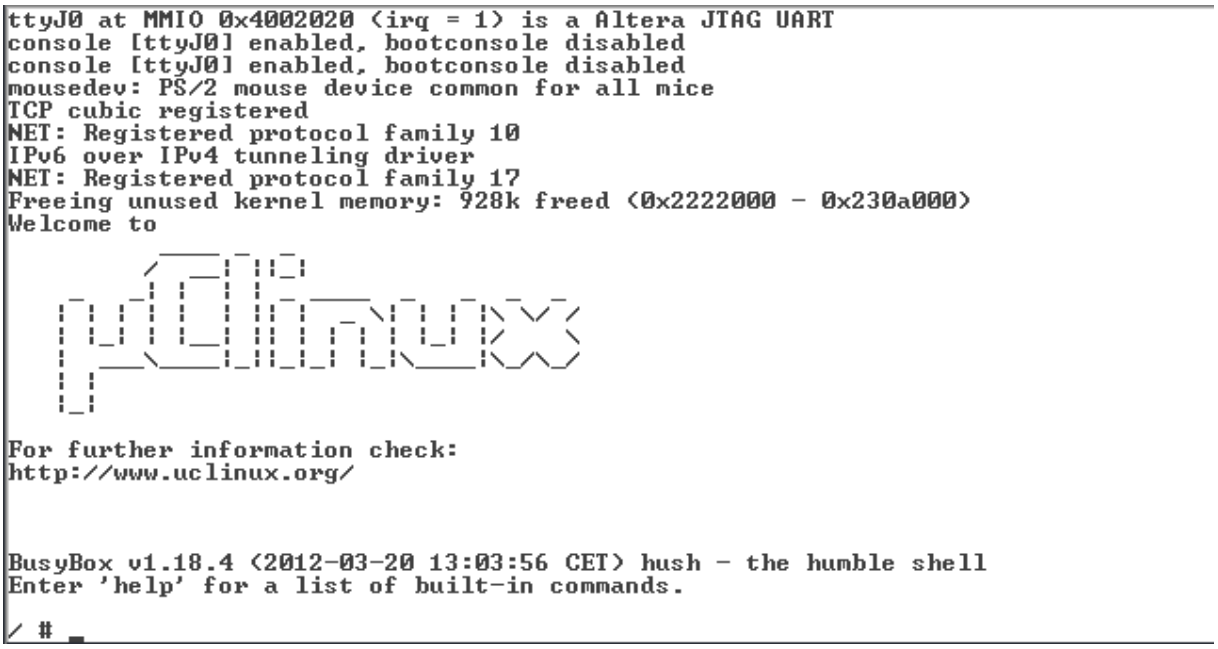
   Pozn. Na počítači, na kterém se provádí kompilování distribuce, je zapotřebí instalace nástroje uboot-mkimage, který je součástí mnoha distribucí, nebo ho lze získat na strákách projektu (http://www.denx.de/wiki/U-Boot/).
8. Pokud se kompilace zdařila, bude výsledný obraz distribuce umístěn v souboru <nios2-linux/uClinux-dist/images/linux.initramfs.gz>. Dalším krokem je zkopírování souboru na počítač, na kterém jsou nainstalovány nástroje Altera, které jsou součástí sofware Quartus.
9. Je potřeba spustit nástroj NIOS II Command Shell, který js součástí software NIOS II Embedded Design Suite (NIOS II EDS).
10. Je třeba nakonfigurovat FPGA pomocí sof souboru:

    />nios2-configure-sof soubor.sof
11. Je třeba nahrát obraz distribuce do paměti RAM:

    />nios2-download –g linux.initramfs.gz
12. Terminál pro připojení k systému lze spusit následujícím způsobem:

    />/nios2-terminal

Poté dojde k zavedení systému, výpis může být podobným výpisu, uvedenému na obr. 6.6. Následně je již možno přes tento terminál zadávat příkazy a pracovat se systémem.



Obr. 6.6 Výpis obrazovky po spuštení systému Linux

6.6 Spuštění systému s podporou MMU

Systém s podporou MMU je potřeba použít tam, kde by mohly uživatelské aplikace přímým přístupem do paměti způsobit pád systému a kde by měl tento pád systému kritický dopad. Nevýhodou takového systému s podporou MMU je nižší výkon, který je způsoben vlivem nepřímého přístupu do paměti a také složitější obvodové řešení procesoru.

6.6.1 Přidání podpory MMU (návrh v SOPC builder)

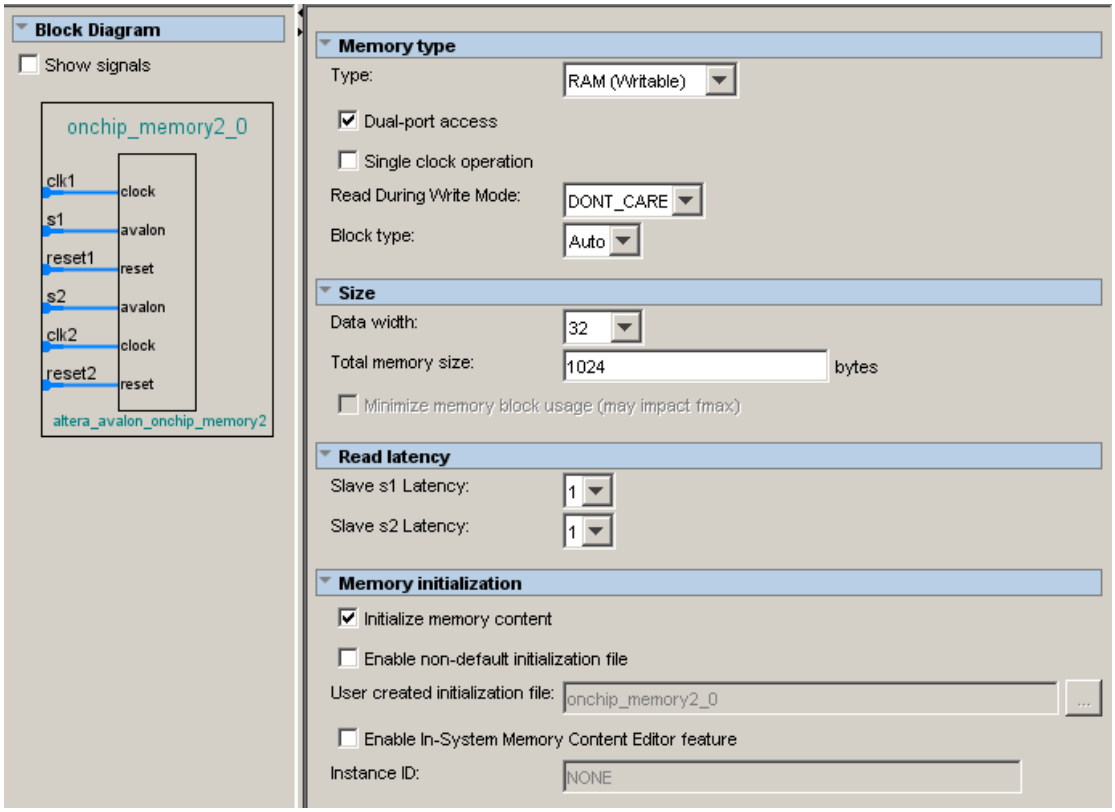
Jak bylo uvedeno v kapitole 1.1.6, může procesor k přímému přístupu do paměti použít pouze rozsah adres 0x000000 – 0x1FFFFFFF, proto je vhodné přiřadit všem komponentám adresy v tomto rozsahu.

Pro zapnutí podpory MMU lze tedy postupovat následovně:

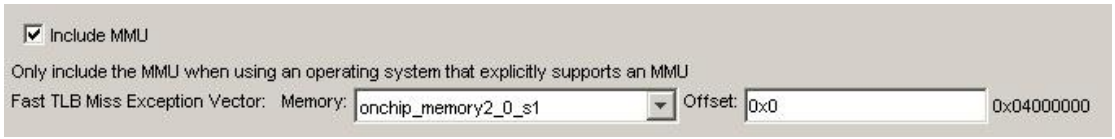
1. Je třeba přidat do projektu v nástroji SOPC Builder, nebo Qsys komponentu *On Chip Memory (RAM or ROM)* se dvěma porty a velikostí 512 b nebo 1024 b. Přesné nastavení je uvedeno na obrázku 6.7.

2. Je nutné otevřít okno pro konfiguraci procesoru a zaškrtnout volbu *Include MMU* a *exception* vektor nastavit na nově vytvořenou *On Chip Memory*. V záložce *Caches and Memory Interface* je třeba zaškrtnout volbu *Include tightly coupled instruction*

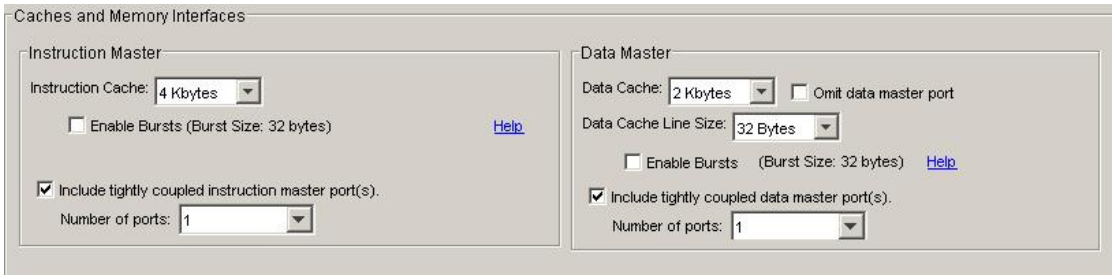
master port(s) a volbu *Include tightly coupled master port(s)*. Počet portů je třeba nechat na výchozí nastavené hodnotě (jedna). Předchozí dvě operace jsou znázorněny na obrázku 6.8 a 6.9.



Obr. 6.7 Přidání podpory MMU, SOPC Builder, krok 1

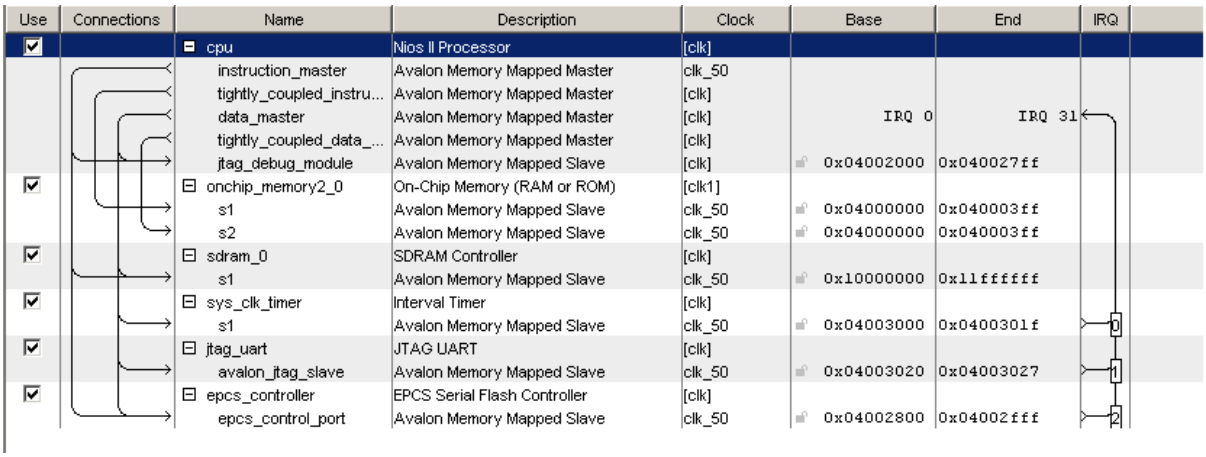


Obr. 6.8 Přidání podpory MMU, SOPC Builder, krok 2



Obr. 6.9 Přidání podpory MMU, SOPC Builder, krok 3

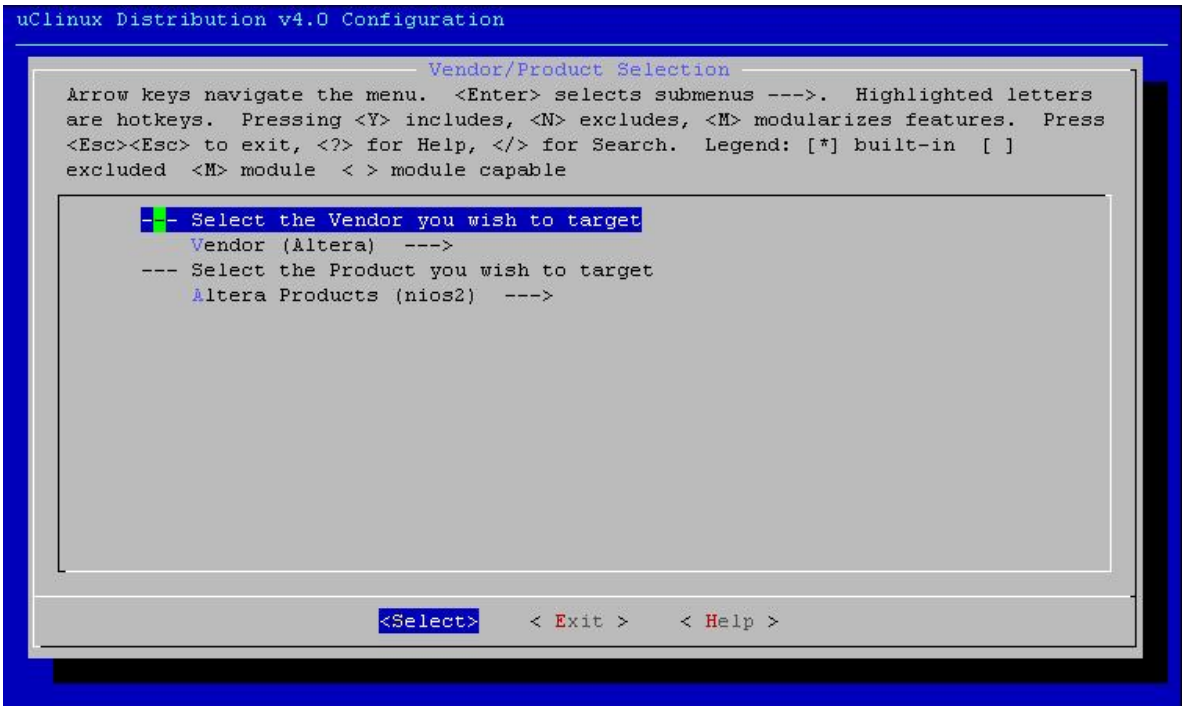
Následující obrázek ukazuje příklad, jak může konečný návrh s podporou MMU vypadat.



Use	Connections	Name	Description	Clock	Base	End	IRQ
☑		cpu	Nios II Processor	[clk]			
		instruction_master	Avalon Memory Mapped Master	clk_50			
		tightly_coupled_instru...	Avalon Memory Mapped Master	[clk]			
		data_master	Avalon Memory Mapped Master	[clk]			IRQ 0
		tightly_coupled_data_...	Avalon Memory Mapped Master	[clk]			IRQ 31
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x04002000	0x040027ff	
☑		onchip_memory2_0	On-Chip Memory (RAM or ROM)	[clk1]			
		s1	Avalon Memory Mapped Slave	clk_50	0x04000000	0x040003ff	
		s2	Avalon Memory Mapped Slave	clk_50	0x04000000	0x040003ff	
☑		sdram_0	SDRAM Controller	[clk]			
		s1	Avalon Memory Mapped Slave	clk_50	0x10000000	0x11ffffff	
☑		sys_clk_timer	Interval Timer	[clk]			
		s1	Avalon Memory Mapped Slave	clk_50	0x04003000	0x0400301f	0
☑		jtag_uart	JTAG UART	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_50	0x04003020	0x04003027	1
☑		epcs_controller	EPCS Serial Flash Controller	[clk]			
		epcs_control_port	Avalon Memory Mapped Slave	clk_50	0x04002800	0x04002fff	2

Obr. 6.10 Návrh systému s podporou MMU, SOPC Builder

Kompilace a spuštění distribuce operačního systému Linux je stejná, jako v předchozí kapitole. Rozdíll je pouze v tom, že v konfiguraci distribuce je potřeba vybrat místo „nios2mmu“ volbu „nios2“ tak, jak ukazuje následující obrázek.



Obr. 6.11 Konfigurace distribuce pro systém s podporou MMU

Po startu systému se lze o podpoře MMU přesvědčit v souboru </proc/cpuinfo> zadáním příkazu />**cat /proc/cpuinfo**

Obr. 6.12 Výpis informací o procesoru s pordporou MMU

6.7 Zabezpečení systému pomocí hesla

Ve výchozím nastavení distribuce dojde po zavedení systému k automatickému přihlášení uživatele root, aniž by byl uživatel dotázán na heslo. Kdokoliv se tak může do systému přihlásit přes sériový port nebo rozhraní JTAG UART. K nastavení systému tak, aby se před přihlášením do systému dotazoval na uživatelské jméno a heslo, je nejprve potřeba přidat do systému několik nástrojů z balíku Busybox. V konfigurační nabídce distribuce *Customize Application/Library Settings* je nutno přidat volby uvedené níže a opět zkompilovat distribuci.

Záložka BusyBox → Init Utilities

[*] *init*

[*] Support reading an inittab file

[ ] Support killing processes that have been removed from inittab

[*] Run commands with leading dash with controlling tty

[*] Enable init to write to syslog

[*] Be _extra_ quiet on boot

[*] Support dumping core for child processes (debugging only)

[*] Support running init from within an initrd (not initramfs)

(linux) Initial terminal type

Záložka BusyBox → Login/Password Management Utilities

[*] getty
[*] login
[*] passwd

Poté je nutné přidat inicializační řádek do souboru </etc/inittab>. Pokud dochází k sestavení systému jako jednoho výsledného obrazu, je možno tento soubor před kompilací upravit v umístění <nios2-linux/uClinux-dist/vendors/Altera/nios2/inittab>. Tento inicializační řádek může vypadat následovně [30]:

```
::respawn:/sbin/getty -L ttyJ0 115200 vt100
```

Uvedený příklad platí pouze v případě použití rozhraní JTAG UART. Jedná-li se o jiné rozhraní, je nutné zaměnit `ttyJ0`, za označení daného zařízení.

Po zavedení systému se objeví na příkazové řádce dialog pro přihlášení do systému. Ve výchozím nastavení distribuce není pro uživatele root nastaveno žádné heslo. Heslo lze nastavit po prvním přihlášení do systému příkazem />**passwd**.

Heslo lze také nastavit a přidat přímo do obrazu distribuce. Pro vygenerování hesla lze využít 3 metody:

1. Pomocí nástroje MKPASSWD, pokud je k dispozici vygenerovat heslo pro uživatele root />**mkpasswd uClinux –H md5** `uClinux` je heslo, které bude šifrováno.
   Vygenerované zašifrované heslo by mělo mít následující tvar:
   `$1$TQHH7TPp$EO30ubQdjCewHPj.oWMtZ.`
2. Na počítači s operačním systémem Linux vytvořit nového uživatele a nastavit heslo: />**useradd tmp_user && passwd tmp_user**. Následně je potřeba zadat dvakrát stejné heslo pro ověření. Zadané šifrované heslo lze poté v systému nalézt nakonci souboru </etc/passwd>. Řádek má následující tvar:
   ```
   tmp_user:$1$TQHH7TPp$EO30ubQdjCewHPj.oWMtZ.:0:0:tmp_user:/home/tmp_
   user:/bin/sh
   ```
   Heslo lze vždy nalézt za uživatelským jménem, v uvedeném případě je tedy heslo:
   `$1$TQHH7TPp$EO30ubQdjCewHPj.oWMtZ.`
3. Spustit distribuci NIOS II Comunnity edition, pomocí nástroje PASSWD změnit heslo uživatele root a zkopírovat ho ze souboru </etc/passwd> obdobně jako v případě druhé metody.

Takto získané heslo je nutné uložit do souboru <nios2-linux/uClinux-dist/romfs/etc/passwd>. Výsledný řádek v tomto souboru může vypadat následovně:

```
root:$1$TQHH7TPp$EO30ubQdjCewHPj.oWMtZ.:0:0:root:/:/bin/sh
```

Po úpravě tohto souboru je potřeba spustit v adresáři <nios2-linux/uClinux-dist> příkaz />**make image**. Spuštěním příkazu dojde k sestavení nového obrazu distribuce, který bude obsahovat upravený soubor **passwd**.

6.8 Síťová karta DM9000

Vývojová deska DE2-70 obsahuje síťovou kartu s čipem DM 9000 od výrobce Davicom semiconductor. Čip disponuje 16 KB pamětí SRAM a splňuje standard pro 100 Mbit/s Ethernet IEEE 802.3u a dále standard IEEE 802.3x pro plně duplexní režim s řízením toku dat. Ovladače pro tento čip pro operační systém Linux zveřejnila společnost Davicom již v roce 1997 pod licencí GNU. Od té doby jsou obsaženy ve všech verzích jádra.

Komponentu pro nástroj SOPC Builder lze nalézt v příkladech pro vývojový kit nebo na přiloženém DVD ve složce <components>. Komponentu stačí přidat do systému běžným způsobem a propojit piny dle manuálu k vývojové desce DE2-70.

Ovladač pro operační systém Linux dodnes nepodporuje konfiguraci pomocí device tree struktury, proto je nutno provést konfiguraci ručně. Konfigurace bude provedena podle dokumentace k ovladači, kterou obsahuje archiv se zdrojovými kódy jádra. U distribuce NIOS II Linux Community Edition se dokumentace nachází v souboru <uClinux-dist/linux-2.6.x/Documentation/networking/dm9000.txt>. Konfiguraci je potřeba zapsat na konec souboru <linux-2.6/arch/nios2/kernel/setup.c>. Příklad je uveden v příloze A. Dále je uveden postup, jakým způsobem přidat do jádra ovladač pro čip DM9000.

1. Nejprve je nutné Spustit konfiguraci jádra a přidat podporu pro DM9000. Ovladač lze zkompilovat přímo do jádra nebo jako modul jádra.

   Záložka Device Drivers → Network device support → Ethernet driver support
   <*> DM9000 Support

   Pozn.: Pokud tato volba chybí (jádro od verze 3), je potřeba upravit konfigurační soubor <linux-2.6/drivers/net/ethernet/davicom/Kconfig> tak, aby se při konfiguraci jádra volba zobrazila pro architekturu NIOS II. Správné nastavení tohoto souboru je uvedeno v příloze B.

Záložka Device Drivers → PHY Device Support and infrastructure

<*> Drivers for Davicom PHYs

2. V případě, kdy je nutné načíst IP adresu z DHCP serveru. Je dále zapotřebí zkompilovat nástroj DHCPDC, který je součástí distribuce. V konfigurační nabídce aplikací je potřeba vybrat volbu:

   Záložka Network Applications

   [*] dhcpcd-new (2.0/2.4)

Po přeložení jádra a načtení systému se ve výpisu jádra objeví zpráva podobná následující:

```
dm9000 Ethernet Driver, V1.31
eth0: dm9000a at 84001028,8400102c IRQ 2 MAC:
00:01:02:03:04:05 (chip)
```

Z této zprávy lze vyčíst, že k adaptéru bylo přiřazeno označení eth0 a byla nastavena výchozí MAC adresa 00:01:02:03:04:05. Pokud byl ovladač zkompilován jako modul jádra, lze ho zavést příkazem />**modprobe dm9000**.

Ke konfiguraci síťového zařízení lze využít příkaz />**ifconfig**. Pro změnu MAC adresy />**ifconfig eth0 hw ether xx:xx:xx:xx:xx:xx,** (xx:xx:xx:xx:xx:xx je nutné nahradit konkrétní adresou). Pro nastavení pevné IP adresy lze použít zápis: />**ifconfig eth0 192.169.1.1 up**. Pro nastavení IP adresy z DHCP serveru zápis: />**ifconfig eth0 up && dhcpcd &**

Pozn.: Všechna výše popsaná řešení fungují jak pro systém s podporou MMU, tak i pro systém bez podpory MMU.

6.8.1 Přenosová rychlost a odezva použitého síťového adaptéru DM9000A

K testování použitého síťového adaptéru, který obsahuje vývojová deska DE2-70, tedy adaptéru s čipem DM9000A, byly použity jednoduché testy. Pomocí nástroje PING byla testována odezva a pomocí nástroje WGET rychlost adaptéru. Tyto tesy je tedy nutné brát pouze orientačně.

Testy byly prováděny na počítači DELL se síťovým adaptérem Intel 82567LM 10/100/1000Mbit a SSD diskem. Použit byl běžný dvoumetrový UTP kabel. Pro získání referenčních hodnot bylo měření prováděno proti počítači se síťovým adaptérem BCM440x 10/100Mbit.

Pro měření rychlosti adaptéru byl použit http server Apache verze 2.2 a soubor o velikosti 10 MB. Pro test rychlosti odezvy byl použit nástroj PING s počtem opakování 100. Testy byly provedeny na jádře NIOS II/f a systému s podporou MMU. V následující tabulce jsou uvedeny výsledky tohoto testu.

	Ping [ms]			
	Maximální	Minimální	Průměrná	Přenosová rychlost [Mb/s]
NIOS II/f, DM 9000A	1,976	1,263	1,568	1,371
Referenční hodnota (běžný PC se 100Mbit síťovým adaptérem)	0,264	0,123	0,201	38,2

Tab. 6.1 Rychlost a odezva adaptéru DM9000A

Jak je možné vidět z tabulky 6.1, není odezva, ani přenosová rychlost adaptéru s čipem 9000A příliš velká. To je způsobeno zejména čistě softwarovou obsluhou a absencí hardwarové akcelerace. Další snížení rychlosti mohlo být způsobeno vlastním výkonem procesoru NIOS II. Je nutné si uvědomit, že podle tesů rychlosti zápisu do paměti (viz kapitola 9.3.3) je u této konfigurace systému rychlost zápisu do paměti pouze 7,63 MB/s.

6.8.2 Využití síťového adaptéru

Obecně lze říci, že v operačních systémech Linux existuje velké množství síťových serverů a služeb (např. servery typu HTTP, FTP, NFS, VPN a další) nebo aplikací pro komunikaci (jako jsou *chat*, e-mail, atd.). I v distribucích operačního systému Linux pro embedded systémy lze najít mnoho takovýchto aplikací a serverů, které jsou mnohdy uzpůsobené pro běh na procesorech s omezeným výkonem a omezenou operační pamětí. Cílem této kapitoly

je demonstrovat možnosti využití síťové komunikace u embedded systému na známých serverech a službách. Vývoj těchto aplikací probíhá velmi rychle a jejich počet se stále zvyšuje. Z těchto důvodů je dobré pro konkrétní účely nejprve zjistit možnosti všech řešení, která jsou aktuálně dostupná a vybrat to, které bude nejvíce vyhovovat dané aplikaci.

6.8.2.1 Připojení vzdáleného adresáře, protokol NFS (*Network file system*)

Jak již název napovídá, NFS je protokol pro sdílení souborů přes počítačovou síť. Zejména v prostředí opračního systému Linux se jedná o nejpoužívanější způsob, jak připojit adresář ze vzdáleného serveru a pracovat s ním [31].

Pro použití NFS klienta je potřeba v konfiguraci jádra zaškrtnout následující volby

Záložka File systems → Network File Systems
<*> NFS Client support
[*] NFS Client support for NFS version 3
[*] NFS Client support for the NFSv3 ACL protocol extension
Dále v konfiguračním menu aplikací a knihoven volby:
Záložka Network Applications
[*] portmap

Připojení vzdáleného souborového systému se poté provede následovně:

/>portmap
/>mkdir /mnt/nfs
/>mount -t nfs -n -o nolock,rsize=1024,wsize=1024 ipaddr:/rmtdir locdir

V posledním příkazu je nutné nahradit `ipaddr` za IP adresu vzdáleného serveru, `rmtdir` absolutní adresou k adresáři na vzdáleném serveru a `locdir` cestou k adresáři na lokálním zařízení, do kterého bude vzdálený adresář připojen.

V případě častého připojování vzdáleného adresáře, lze celou věc usnadnit tím, že se nadefinuje zdrojové i cílové místo připojení a parametry souborového systému. To lze zajistit prostřednictvím souboru **fstab**. Tento soubor lze přidat do výsledného obrazu distribuce, který je umístěn do adresáře <uClinux-dist/romfs/etc/>. Zápis pro připojení vzdáleného adresáře může v tomto souboru vypadat následovně:

```
10.10.110.99:/home/fandar /mnt/nfs nfs rsize=1024,
wsize=1024,noauto,nolock 0 0
```

`10.10.110.99:/home/fandar` je adresa s adresářem na vzdáleném NFS serveru a `/mnt/nfs` cílovým místem připojení. Tímto způsobem lze nadefinovat veškeré NFS servery a adresáře, se kterými se bude v systému pracovat. Připojení se poté provede příkazem />**mount /mnt/nfs**.

Celou věc lze navíc věc zautomatizovat tak, aby k připojení síťových disků došlo automaticky po spuštění operačního systému Linux. K tomu lze využít inicializační soubor <nios2-linux/uClinux-dist/vendors/Altera/common/rc>. Níže je uveden příklad, jak může zápis pro konfiguraci sítě a připojení síťového disku vypadat:

```
ifconfig eth0 hw ether 00:0b:ff:ee:cc:aa
ifconfig eth0 10.10.110.243 up
mkdir /mnt/nfs
mount /mnt/nfs
```

V případě, že je třeba pouze pozměnit inicializační soubor a není nutné kompilovat celou distribuci, lze změný inicializační souboru provést v umístění <nios2-linux/uClinux-dist/romfs/etc/rc> a vytvořit nový obraz distribuce příkazem />**make image**.

6.8.2.2 Připojení kořenového adresáře přes protokol NFS (NFS rootfs)

Jádro operačního systému Linux umožňuje po zavedení základního systému připojit kořenový adresář přes protokol NFS, to znamená, že veškeré aplikace, knihovny a ostatní soubory, které používají uživatelé systému, lze hostovat na vzdáleném serveru. Výhodou tohoto řešení je značná úspora kapacity paměti embedded systému a ukládání všech souborů na vzdálený server, kde lze snadněji provádět jejich zálohu. Kořenový adresář lze navíc sdílet s jinými systémy. Nevýhodou může být naopak zpomalení běhu systému v důsledku síťové komunikace se vzdáleným strojem.

Po zkompilování jádra, které lze provést podle jednoho z předchozího způsobu, by se měla struktura kořenového adresáře nalézat ve složce <nios2-linux/uClinux-dist/romfs/>. Pokud nechceme připojovat přímo tento adresář, např. z důvodu časté kompilace a testování distribuce, lze jeho obsah zkopírovat do jiného umístění, např. </home/nios2>. />**cp -R uClinux-dist/romfs/* /home/nios2**. Tento adresář </home/nios2> je dále potřeba zpřístupnit pro vzdálený systém pomocí protokolu NFS a standardním způsobem spustit operační systém Linux na vývojové desce.

Dále je potřeba připojit vzdálený adresář a zkopírovat do něj složku </dev>. />**cp -a /dev/* /mnt/nfs/dev**. Poté je kořenový adresář připraven k použití a je možno změnit konfiguraci jádra tak, aby došlo k načtení kořenového souborového systému přes NFS protokol. K tomu Je nutno aktivovat následující volby:

Záložka Networking → Networking Options

[*] TCP/IP networking

[*] IP: kernel level auto configuration

[*] IP: DHCP support

Záložka File systems → Network File Systems

[*] Root file system on NFS

Dále je potřeba zajistit, aby po zavedení jádra systému došlo ke konfiguraci síťového adaptéru a k připojení kořenového adresáře přes NFS protokol. To lze udělat pomocí volby *command string* v konfigurační nabídce jádra [32].

Záložka Processor type and features

() Default kernel command string

Nastavení této volby pro statickou IP adresu může vypadat následovně

```
root=/dev/nfs rw nfsroot=10.10.110.99:/home/nios2
ip=10.10.110.243:10.10.110.9:10.10.110.1:255.255.255.0:nios2:eth0:off
```

`10.10.110.9` je IP adresa stroje, na kterém běží NFS server, `/home/nios2` je cesta k adresáři na vzdáleném serveru, `10.10.110.243` je IP adresa pro vývojovou desku, `10.10.10.1` označuje bránu a `255.255.255.0` je maska sítě. Parametr `nios2` je síťové jméno pro náš systém a `eth0` označení pro síťové zařízení.

V případě nastavení adresy z dhcp serveru může vypadat volba *command string* následovně

```
root=/dev/nfs rw nfsroot=10.10.110.99:/home/nios2 ip=dhcp
```

`10.10.110.99` je adresa stroje, na kterém běží NFS server a `/home/nios2` je cesta k adresáři na vzdáleném serveru.

6.8.2.3 Webový server

Na internetu lze najít mnoho webových serverů, s velikostí pouze několik KB, které jsou vhodné pro embedded systémy. Základní distribuce uClinux obsahuje hned několik těchto

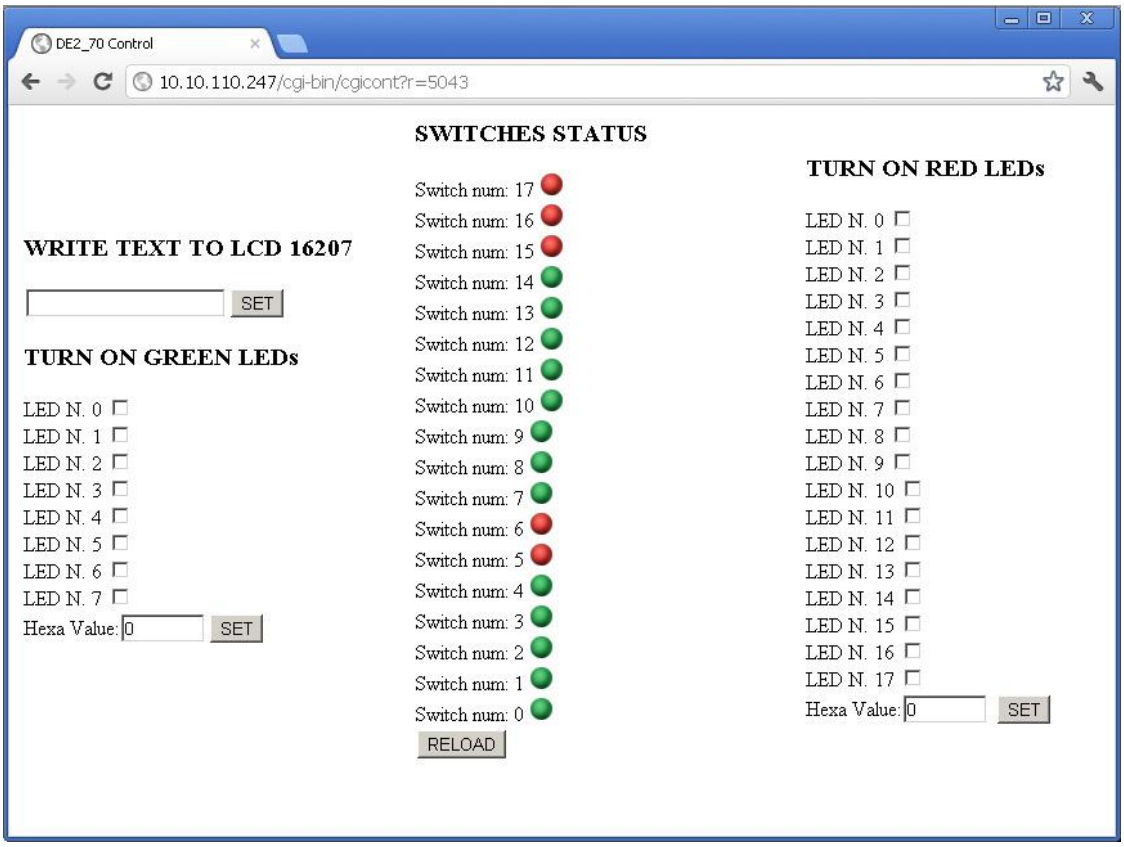
serverů. Mezi nejznámější patří webový server BOA, Thttpd (*tiny/turbo/throttling HTTP server*) a appWeb. Všechny tyto servery podporují spouštění CGI skriptů, dále bude jako příklad uvedena práce se serverem Thttpd, který umožňuje mimo jiné jednoduchým způsobem zabezpečit obsah uživatelským jménem a heslem.

Autorem webového serveru Thttpd je pan Jeff Poskanzer. Jak sám autor na svých stránkách uvádí, bylo jeho záměrem vytvořit malý, jednoduchý, rychlý a přenositelný webový server. Projekt vznikl v roce 1995 a je volně šířen pod licencí BSD. Je nutno uvést, že Thttpd nemá implementován kompletní standard HTTP 1.1, ale pouze jeho nutné minimum. Právě to umožnilo podstatným způsobem zredukovat výslednou velikost celého serveru [33].

Server Thttpd je možno do distribuce přidat zaškrtnutím patřičné volby v konfigurační nabídce aplikací a knihoven. Thttpd je možno spustit následujícím příkazem. Pokud chceme, aby ke spuštění docházelo automaticky po startu systému, lze tento příkaz zapsat do inicializačního souboru *rc*. />**/bin/thttpd -d /home/httpd -u root -c '/cgi-bin/*' -p 80 &**
Příznak `-d` udává kořenový adresář pro stránky, příznak `-c` představuje složku s CGI skripty a příznak `-p` udává port, na kterém bude server naslouchat. Výpis ostatních možných voleb lze získat zadáním příkazu />**/bin/thttpd --help.** Parametry je také možno specifikovat v konfiguračním souboru. Více informací lze poté nalézt v dokumentaci k serveru.

Pro zabezpečení obsahu je možno použít protokol *basic access authentication* (jednoduché ověření přístupu). Toto zabezpečení funguje tak, že webový server pošle dotaz pomocí protokolu HTTP na uživatelské jméno a heslo. U serveru Thttpd není potřeba provádět žádnou speciální konfiguraci pro využití tohoto druhu zabezpečení. Stačí umístit soubor **.htpasswd** do adresáře, jehož obsah má být tímto způsobem chráněn. Tento soubor s uživatelskými jmény a hesly lze vytvořit pomocí programu HTPASSWD na stroji s operačním systémem Linux.

Jako ukázka příkladu využití webového serveru byl v rámci této práce napsán CGI skript (viz obr 6.13), který umožňuje ovládat LED diody, LCD displej 16207 a zobrazit polohu přepínačů. Zdrojový kód tohoto skriptu je uveden v příloze G.



Obr. 6.13 CGI skript pro ovládání LED, LCD a přepínačů

6.8.2.4 Vzdálená komunikace (SSH server)

Pro vzdálenou komunikaci se zařízením s operačním systémem Linux lze využít protokol SSH. Distribuce uClinux obsahuje nástroj DROPBEAR, který v jediném binárním souboru obsahuje SSH server, SSH klienta a SCP klienta pro bezpečné kopírování souborů ze vzdáleného systému. Zkompilovaný binární soubor má velikost kolem 1,2 MB a podporuje protokol SSH 2 [34].

Po zkompilování nástroje je binární soubor s názvem **dropbearmulti** umístěn v adresáři </bin>. V adresáři </etc/dropbear> se nalézají zabezpečující klíče pro ssh server, popřípadě konfigurační soubory. Jak již bylo uvedeno, tento binární soubor obsahuje hned několik nástrojů. K jejich dalšímu využití je potřeba vytvořit tzv. symbolické odkazy:

/>ln –s /bin/dropbearmulti /bin/dropbear pro ssh server

/>ln –s /bin/dropbearmulti /bin/ssh pro ssh klienta

/>ln –s /bin/dropbearmulti /bin/scp pro scp klienta

Po spuštění serveru příkazem />**/bin/dropbear** se lze poté k systému připojit přes protokol SSH. Server standardně naslouchá na portu 22.

6.9 LCD display 16207

Pro podporu tohoto displeje je možné vycházet z ovladače, jehož zdrojové kódy lze stáhnout na stránkách projektu altera wiki [24]. Jedná se o ovladač typu znakového zařízení. Výchozí hlavní číslo pro toto zařízení je nastaveno na 250 a soubor zařízení je </dev/lcd16207>. Tento originální ovladač však nefunguje s novou verzí jádra (od verze 3.0.0 a vyšší) a pracuje pouze na systému bez podpory jednotky správy paměti. Na systému s podporou MMU je zapotřebí navíc provést „přemapování“ fyzické adresy na virtuální. Úpravy verze původního ovladače pro novou verzi jádra a pro systém s podporou MMU jsou uvedeny v příloze C a D. Pro systém bez MMU jsou úpravy ovladače uvedeny v přílohách E a F. Úpravy jsou výstupem nástroje DIFF a k jejich rychlému aplikování je možno využít v prostředí operačního systému Linux nástroj PATCH.

V souboru **lcd_16207.h** je dále nutno specifikovat adresu (*base address*), která je definována při návrhu systému. Soubory **lcd_16207.h** a **lcd_16207.c** je poté nutno nakopírovat do složky <uClinux-dist/linux-2.6.x/source/drivers/char/> a přidat do konfigurační nabídky jádra novou volbu pro tento ovladač (viz kapitola 6.2).

Po zkompilování a spuštění jádra s tímto ovladačem by se na displeji objevý text „Hello“. Dále je možno na displej zapisovat jednoduchým způsobem pomocí příkazu ECHO (např. />**echo “Ahoj” > /dev/lcd16207**).

6.10 Podpora pro zařízení USB Mass Storage

Vývojový kit DE2-70 obsahuje řadič USB od firmy PHILIPS s označením ISP1362. Ovladače pro tento typ jsou obsaženy již v jádře operačního systému Linux a fungují jak v systému s podporou MMU, tak i v systému bez MMU. Je však nutno vytvořit novou komponentu v programu SOPC Builder za použití verilog souboru, který lze opět nalézt na webových stránkách altera wiki [24], nebo na přiloženém DVD. V záložce *interface* je poté nutno provést následující nastavení:

- Slave Adressing: Native

- Units: ns
- Setup: 100
- Read Wait: 100
- Write Wait: 100
- Hold: 100

Po přidání této komponenty do projektu je nutno ji přejmenovat na ISP1362. Dále je potřeba v projektu přiřadit vhodným způsobem piny.

Následně je nutné vhodně zkonfigurovat jádro systému a přidat podporu pro ovladač a *USB Mass Storage*, aktivováním následujících voleb:

Záložka Device Drivers → SCSI device support
[*] SCSI device support
[*] legacy /proc/scsi/ support
[*] SCSI disk support
Záložka Device Drivers → USB support
[*] Support for Host-side USB
[*] USB verbose debug messages
[*] USB announce new device
[*] USB device filesystem
[*] USB device class-devices
[*] ISP1362 HCD support
[*] USB Mass Storage support
Popřípadě lze v této záložce přidat podporu pro jiné USB zařízení.

Dále je třeba přidat podporu souborového systému, který používá USB zařízení a vhodnou kódovou sadu. Níže je uveden příklad nastavení pro souborový systém FAT a kódovou sadu pro americké a středoevropské znaky.

Záložka File systems → DOS/FAT/NT Filesystems
[*] VFAT (Windows-95) fs support
(437) Default codepage for FAT
(iso8859-1) Default iocharset for FAT
Záložka File systems → Native language support
[*] Codepage 437 (United States, Canda)

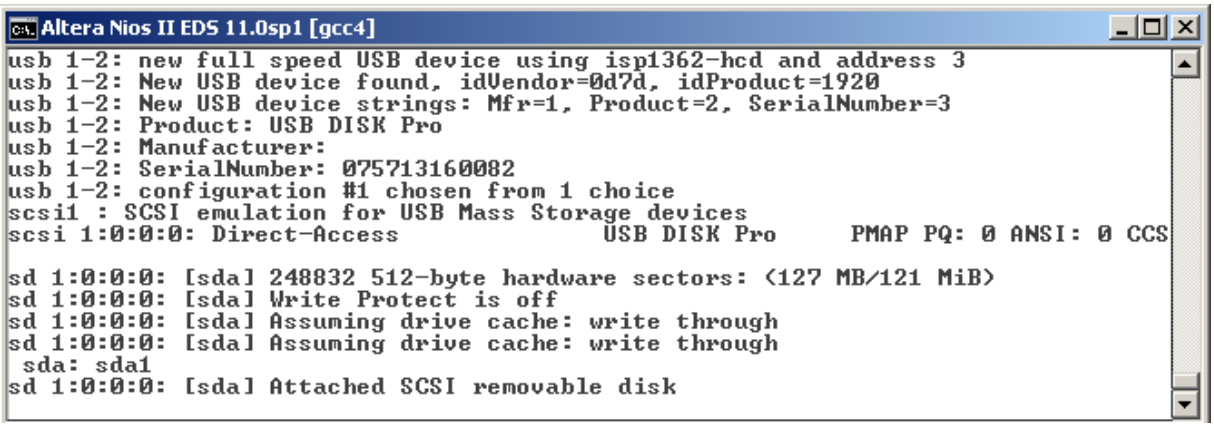
[*] Codepage 852 (Central/Eastern Europe)

Před zkompilováním jádra je nutné upravit originální ovladač **isp1362-hcd.c**. Originální ovladač při spuštění vypisuje chybu, že není zařízení připraveno a po vložení USB zařízení, nedojde k jeho registrování do systému. Úprava ovladače spočívá v editaci zdrojového kódu </linux-2.6/drivers/usb/host/isp1362-hcd.c> následovně:

Funkci `static int isp1362_hc_reset(struct usb_hcd *hcd)` je potřeba nahradit kódem:

```
static int isp1362_hc_reset(struct usb_hcd *hcd)
{
	int ret = 1;
	struct isp1362_hcd *isp1362_hcd = hcd_to_isp1362_hcd(hcd);
	return ret;
}
```

Po připojení zařízení s podporou *USB Mass Storage* jádro systému upozorní na připojení nového zařízení a zobrazí informace, které jsou o daném zařízení k dispozici.



Obr. 6.14 Výpis jádra po připojení nového zařízení s podporou USB Mass Storage

Z výpisu na obr. 6.14 je patrné, že bylo registrováno nové zařízení s názvem *sda*, pro připojení daného souborového systému a je možno využít linuxový nástroj MOUNT. Postup je následující:

1. Nejprve je nutné vytvořit adresář po připojení: />**mkdir /mnt/usb**.
2. Je třeba připojit oddíl do daného umístění: />**mount /dev/sda1 /mnt/usb**, `sda1` je označení oddílu, pokud je k dispozici nástroj FDISK, lze informace o oddílech získat příkazem />**fdisk –l /dev/sda**.

6.11 Ovládání LED diod

V případě systému bez podpory MMU, lze k periferiím přistupovat velice jednoduše a to přímou adresací v paměti. V tomto jednoduchém příkladu je popsáno, jak rozsvítit nebo zhasnout zelené LED diody. Deska DE2-70 obsahuje celkem osm těchto diod. Všechny LED diody se rozsvěcují přivedením hodnoty logické „1“. Přesné zapojení je možné nalézt v manuálu k vývojové desce [25].

Pro práci s diodami je potřeba do projektu v nástroji SOPC builder, popřípadě Qsys přidat novou komponentu PIO (Parallel I/O). *Width* (šířku) je třeba nastavit podle počtu diod, v tomto případě tedy 8 bitů, a v záložce *Direction* (směr) je třeba zaškrtnout volbu *Outputs port only* (Pouze výstupní porty). Takto přidanou komponentu je možno v tomto případě pojmenovat zcela libovolně (např. green_led). Dále jí dle manuálu vhodným způsobem propojit s piny diod.

Jako ukázku ovládání diod lze vytvořit např. jednoduchý program v jazyce C, který bude v zadaném intervalu rozsvěcovat a zhasínat určité LED diody se zadaným počtem cyklů. Výsledný zdrojový kód může vypadat následovně:

```
1: #include <stdio.h>
2: #define green_led (unsigned short*) 0x0940a160 // Adresa v SOPC builder
3: int main(void)
4: {
5:   int i,blinks_n,time;
6:   unsigned short leds;
7:   printf("LEDky, které budou zapnuty (hexa hodnota, napr. FF) \n");
8:   scanf("%2X",&leds);
9:   printf("cas, po ktery budou zapnuty/vypnuty (ms)\n");
10:      scanf("%i",&time);
11:      printf("pocet blinknuti\n");
12:      scanf("%i",&blinks_n);
13:
14:      for (i=0; i<blinks_n; i++)
15:      {
16:        *green_led=leds; //nastav zadanou hodnotu
17:        usleep(1000*time);
18:        *green_led=0; //zhasni vsechny led
19:        usleep(1000*time);
20:      }
21:      return 0;
22:    }
```

Ve druhém řádku je nutno definovat adresu komponenty tak, jak byla nastavena v prostředí SOPC Builder. Program na pozici paměti LED diod zapisuje hexadecimální hodnotu, kterou zadá uživatel, tzn., že např. zadaná hodnota FF rozsvítí všechny LED diody, zatímco hodnota

0F pouze první čtyři diody. Dalšími vstupy programu jsou čas, po který mají být diody zhasnuty nebo rozsvíceny a počet cyklů blikání diod. Program je nutno zkompilovat příkazem />**nios2-linux-gnu-gcc ledmm.c -o ledmm**, nástroj NIOS2-LINUX-GNU-GCC je součástí balíku nástrojů *toolchain*.

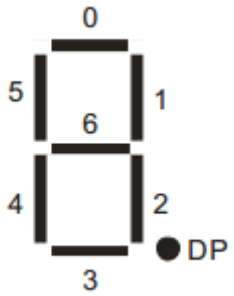
V systému s podporou MMU je potřeba k paměti přistupovat přes soubor </dev/mem> s využitím funkce `mmap`. Modifikovaný kód pro systém s MMU je uveden v příloze H.

6.12 Vytvoření znakového ovladače sedmi-segmentového displeje

Vývojová deska DE2-70 obsahuje celkem osm sedmi-segmentových displejů. Tato kapitola podrobně popisuje, jak vytvořit jednoduchý ovladač pro jeden z těchto displejů. Popis obsahuje návrh systému v programu SOPC Builder, vytvoření znakového ovladače a uživatelské aplikace k jednoduchému ovládání displeje.

Jak je uvedeno v manuálu k vývojovému kitu DE2-70, jsou displeje připojeny přímo k FPGA procesoru. Přivedením logické „0“ na daný segment dojde k jeho rozsvícení, při logické „1“ segment zhasne.

Následující obrázek a tabulka uvádějí očíslování segmentů, označení signálů a rozložení vstupů.



Obr. 6.15 Očíslování segmentů sedmi-segmentového displeje (převzato z [25])

Jméno signálu	**Označení vstupu FPGA**	**Popis**
HEX0_D[0]	PIN_AE8	Displej číslo 0, segment 0
HEX0_D[1]	PIN_AF9	Displej číslo 0, segment 1
HEX0_D[2]	PIN_AH9	Displej číslo 0, segment 2
HEX0_D[3]	PIN_AD10	Displej číslo 0, segment 3
HEX0_D[4]	PIN_AF1	Displej číslo 0, segment 4
HEX0_D[5]	PIN_AD11	Displej číslo 0, segment 5
HEX0_D[6]	PIN_AD1	Displej číslo 0, segment 6
HEX0_DP	PIN_AF1	Displej číslo 0, desetinná čárka

Tab. 6.2 Označení Pinů segmentů sedmi-segmentového displeje

Následující popis obsahuje přidání vhodné komponenty v nástroji SOPC Builder:

1. Nejrpve je nutno přidat novou komponentu PIO (Parallel I/O).
2. Je třeba *width* (šířku) u této komponenty nastavit na 8 bitů.
3. V záložce *direction* (směr) je třeba nastavit: *Outputs port only* (Pouze výstupní porty)
4. Je nutné Koponentě přiřadit vhodnou adresu *Base Address*

Dále je nutno vhodným způsobem přiřadit piny v prostředí Quartus.

Pro vlastní ovladač znakového zařízení je potřeba založit nový soubor s názvem **sedm_seg_disp.c**. K psaní kódu lze využít libovolný textový editor. Nejprve je nutno nadefinovat hlavičkové soubory, které budou dále používány.

```
1: #include <linux/module.h>
2: #include <linux/init.h>
3: #include <linux/kernel.h>
4: #include <linux/ioport.h>
5: #include <linux/fs.h>
6: #include <linux/types.h>
7: #include <asm/io.h>
8: #include <asm/uaccess.h>
```

Dále je vhodné nadefinovat následující makra pro zjednodušení zápisu:
Definování adresy, která byla nastavena v nástroji SOPC Builder, popř. Qsys. V případě systému s podporou MMU, je adresa posunuta o 0xe0000000:

```
9: #define na_seven_seg_1 0x00002450 // 0x00002450 | 0xe0000000 v pripade
   systemu s MMU
```

Makro pro zápis negované hodnoty na displej:

```
10:     #define WRITE_7(val) outl(~((unsigned short) val),
        (na_seven_seg_1))
```

Tzv. „Hlavní číslo“ přiřazené ovladači:

```
11:     #define SEDM_SEG_ MAJOR 70
```

Velikost bufferu:

```
12:     #define BUFFER_SIZE 2
```

Prvotní inicializace (vyprázdnění bufferu a deklarace proměnné pro určení počtu otevřených zařízení):

```
13:     char sedm_seg_disp_buf[BUFFER_SIZE] = "\0";
14:     int sedm_seg_disp_device_open = 0;
```

Dále bude vytvořeno pole, pro určení hexa-kódu podle požadovaného znaku, který se bude zobrazovat na displeji (viz následující tabulka).

Požadovaný znak	hexa kód	binární kód	Rozsvícené segmenty (po negaci)
0	0x3f	0111111	0,1,2,3,4,5
1	0x06	0000110	1,2
2	0x5b	1011011	0,1,3,4,6
3	0x4f	1001111	0,1,2,3,6
4	0x66	1100110	1,2,5,6
5	0x6d	1101101	0,2,3,5,6
6	0x7d	1111101	0,2,3,4,5,6
7	0x07	0000111	0,1,2
8	0x7f	1111111	0,1,2,3,4,5,6
9	0x6f	1101111	0,1,2,3,5,6
A	0x77	1110111	0,1,2,4,5,6
B	0x7c	1111100	2,3,4,5,6
C	0x39	0111001	0,3,4,5
D	0x5e	1011110	1,2,3,4,6
E	0x79	1111001	0,3,4,5,6
F	0x71	1110001	0,4,5,6

Tab. 6.3 Binární a hexa kódy pro zobrazení znaků na displeji

Definované pole vypadá následovně:

```
15:     static unsigned char hex_sedm_seg_data[] = {
16:             0x3F, 0x06, 0x5B, 0x4F, 0x66,
17:             0x6D, 0x7D, 0x07, 0x7F, 0x6F,
18:             0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71
19:     }
```

Funkce pro zapsání hexa-hodnoty na displej:

```
20:     void write_sedm_seg(unsigned char val)
21:     {
```

```
22:             unsigned char seg0;
23:             seg0 = hex_sedm_seg_data[val];
24:
25:     }        WRITE_7(seg0);
```

Odregistrování ovladače:

```
26:     static int sedm_seg_del(struct inode *inode, struct file *file)
27:     {
28:             --sedm_seg_disp_device_open;
29:             module_put(THIS_MODULE);
30:             return 0;
31:     }
```

Otevření zařízení:

```
32:     static int sedm_seg_open(struct inode *inode, struct file *file)
33:     {
34:             //pokud je jiz ovaldac zaregistrovan
35:             if (sedm_seg_disp_device_open++) {
36:                     printk(KERN_ALERT "Device busy...\n");
37:                     return -EBUSY;
38:             }
39:             try_module_get(THIS_MODULE);
40:             return 0;
41:     }
```

Funkce pro získání a kontrolu hodnoty zadané uživatelem:

```
42:     static ssize_t sedm_seg_write(struct file *filp, const char *buff,
   size_t len, loff_t *off)
43:     {
44:             //kontrola velikosti bufferu
45:             if (len > (BUFFER_SIZE-1))
46:                     len = BUFFER_SIZE-1;
47:
48:             // kopirovani hotnot do bufferu
49:             copy_from_user(sedm_seg_disp_buf, buff, len);
50:             sedm_seg_disp_buf[len] = '\0';
51:
52:             // Vystup na segmenty
53:             write_sedm_seg((unsigned char) sedm_seg_disp_buf[0]);
54:             return len;
55:     }
```

Definování operací pro soubor ovladače:

```
56:     static struct file_operations fops_sedm_seg =
57:     {
58:             .write   = sedm_seg_write,
59:             .open    = sedm_seg_open,
60:             .release = sedm_seg_del
61:     }
```

Inicializační modul, který rezervuje místo v paměti a registruje ovladač:

```
62:     static int sedm_seg_init(void)
63:     {
64:             // Rezervovani pameti
65:             if(!request_mem_region((unsigned long) na_seven_seg_1,
   sizeof(np_pio), "sedm_seg"))
66:                     return -1;
67:
68:             // Registerovani ovladace do kernelu
```

```
69:             if(register_chrdev(SEDM_SEG_MAJOR, "sedm_seg_pio",
   &fops_sedm_seg))
70:             {
71:                     release_mem_region((unsigned long) na_seven_seg_1,
   sizeof(np_pio));
72:                     return -EIO;
73:             }
74:
75:             // rozsviceni znaku F pri inicializaci
76:             write_sedm_seg(0x0F);
77:             return 0;
78:     }
```

Odregistrování znakového zařízení:

```
79:     static void sedm_seg_exit(void)
80:     {
81:             unregister_chrdev(SEDM_SEG_MAJOR, "sedm_seg");
82:
83:             // Uvolneni pameti
84:             release_mem_region((unsigned long) na_seven_seg_1,
   sizeof(np_pio));
85:     }
```

Inicializace modulů:

```
86:     module_init(sedm_seg_init);
87:     module_exit(sedm_seg_exit);
```

Tímto je ovladač hotov a lze ho přidat do distribuce. Celý zdrojový kód ovladače je uveden v příloze I. Soubor se zdrojovým kódem je nutno zkopírovat do jednoho z adresářů se zdrojovými kódy ovladačů jádra distribuce, např. <nios2-linux/linux-2.6/drivers/misc/>, ve stejném adresáři je potřeba editovat soubor **Kconfig** a na jeho konec před řádku `endif #MISC_DEVICES` přidat následující:

```
config SEDM_SEG_DISP
   tristate "Ovladac 7seg displeje"
   help
     Ovladac 7seg displeje pro DE2_70.
```

Poté je nutné přidat nakonec souboru **Makefile** řádek:

```
obj-$(CONFIG_SEDM_SEG_DISP) += sedm_seg_disp.o
```

V konfiguraci jádra lze poté přidat tento ovladač:

Záložka *Device Drivers → Misc devices*

[*] *Ovladac 7seg displeje (NEW)*

6.12.1 Vytvoření aplikace pro komunikaci se znakovým zařízením

Pro vlastní aplikaci, která bude prostřednictvím vytvořeného znakového zařízení zapisovat požadovanou hodnotu na sedmi-segmentový displej, bude potřeba vytvořit nový soubor, např. **setseg.c**. Zdrojový kód této aplikace může vypadat následovně:

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #define USAGE() printf("pouziti: ./setseg <hexa hodnota (velke znaky)
   >\n")
4: int main(int argc, char *argv[])
5: {
6: unsigned char v;
7: // kontrola, zda byl zadan parametr
8: if (argc != 2)
9: {
10:     USAGE();
11:     exit(EXIT_FAILURE);
12:     }
13:     // Cteni hexa hodnoty vstupu
14:     if (sscanf(argv[1], "%1X", &v) == 0)
15:     {
16:     USAGE();
17:     exit(EXIT_FAILURE);
18:     }
19:     // Zapis na displej
20:     FILE *sedm_seg_file;
21:     sedm_seg_file = fopen("/dev/sedmseg", "w");
22:     fprintf(sedm_seg_file, "%c", val);
23:     rewind(sedm_seg_file);
24:     fclose(sedm_seg_file);
25:     exit(EXIT_SUCCESS);
26:
27:     }
```

Program kontroluje, zda byl zadán správně parametr (řádek 8) a zda se jedná o hexadecimální číslo 1-F (řádek 14). Dále program otevře soubor ovladače, předá mu zadaný parametr a soubor uzavře (řádek 20-24).

Tento program se zkompiluje v operačním systému Linux za použití nástroje GCC z vhodného balíku nástrojů *toolchian* zadáním příkazu />**nios2-linux-uclibc-gcc -elf2flt setseg.c -o setseg**.

Po spuštění operačního systému Linux na vývojové desce je neprve nutno zaregistrovat soubor ovladače, pomocí nástroje MKNOD />**mknod /dev/sedmseg c 70 0**. `70` určuje hlavní číslo, definované ve zdrojovém kódu ovladače. Poté je již lze nastavit hodnotu na displeji vytvořeným programem SETSEG, např. />**setseg A.**

6.13 Zobrazení grafiky na konzoli, Framebuffer

Framebuffer je abstraktní zařízení, které reprezentuje grafický hardware. Umožňuje aplikacím přístup k fyzickému hardwaru přes přesně definované rozhraní. Programy poté nepotřebují žádné informace o použitém fyzickém zařízení. Framebuffer je nezávislý na velkých systémových knihovnách, jako jsou např. knihovny X Window System, nebo svgalib. Framebuffer mimo jiné umožňuje spustit konzoli v grafickém režimu nebo spustit jeden z okenních systémů [35].

V distribuci Nios 2 Linux Community Edition lze najít ovladače pro Frambuffer, pracující s VGA komponentou z univerzitního programu Altera, kterou lze nalézt na ftp serveru programu (ftp://ftp.altera.com/up/pub/University_Program_IP_Cores/), nebo rovněž na přiloženém DVD ve složce <\components\>. Ne ve všech verzích distribuce tento ovladač funguje bezchybně, proto je otestovaný, funkční zdrojový kód ovladače rovněž dostupný na DVD, které je součástí této práce pod umístěním </drivers/altfb.c>.

Pro tuto komponentu a ovladač zatím chybí podpora device tree struktury. Proto je nutno definovat adresu zařízení, popřípadě také změnit rozlišení a barevnou hloubku zobrazení v souboru ovladače. Jeho umístění v ditribuci je <nios2-linux/linux-2.6/drivers/video/altfb.c>.
Pro podporu systému Framebuffer je v konfiguraci jádra nutno aktivovat následující volby:

Záložka Device Drivers → Graphics support → Support for frame buffer device
<*> Altera framebuffer support
Záložka Device Drivers → Graphics support → Console display driver support
[] VGA text console
< > Framebuffer Console support
Záložka Device Drivers → Character devices
[*] Virtual terminal
[*] Enable character translations in konsole
[] Support for console on virval terminal
Pro podprou USB myši a klávesnice
Záložka Device Drivers → Input device support
<*> Mouse Interface
[*] Provide legacy /dev/psaux device
(640) Horizontal screen resolution

(480) Vertical screen resolution

<*> Event interface

[*] Keyboards

[*] *Mice*

Záložka Device Drivers → HID Devices

<*> USB Human Interface Device (full HID) support

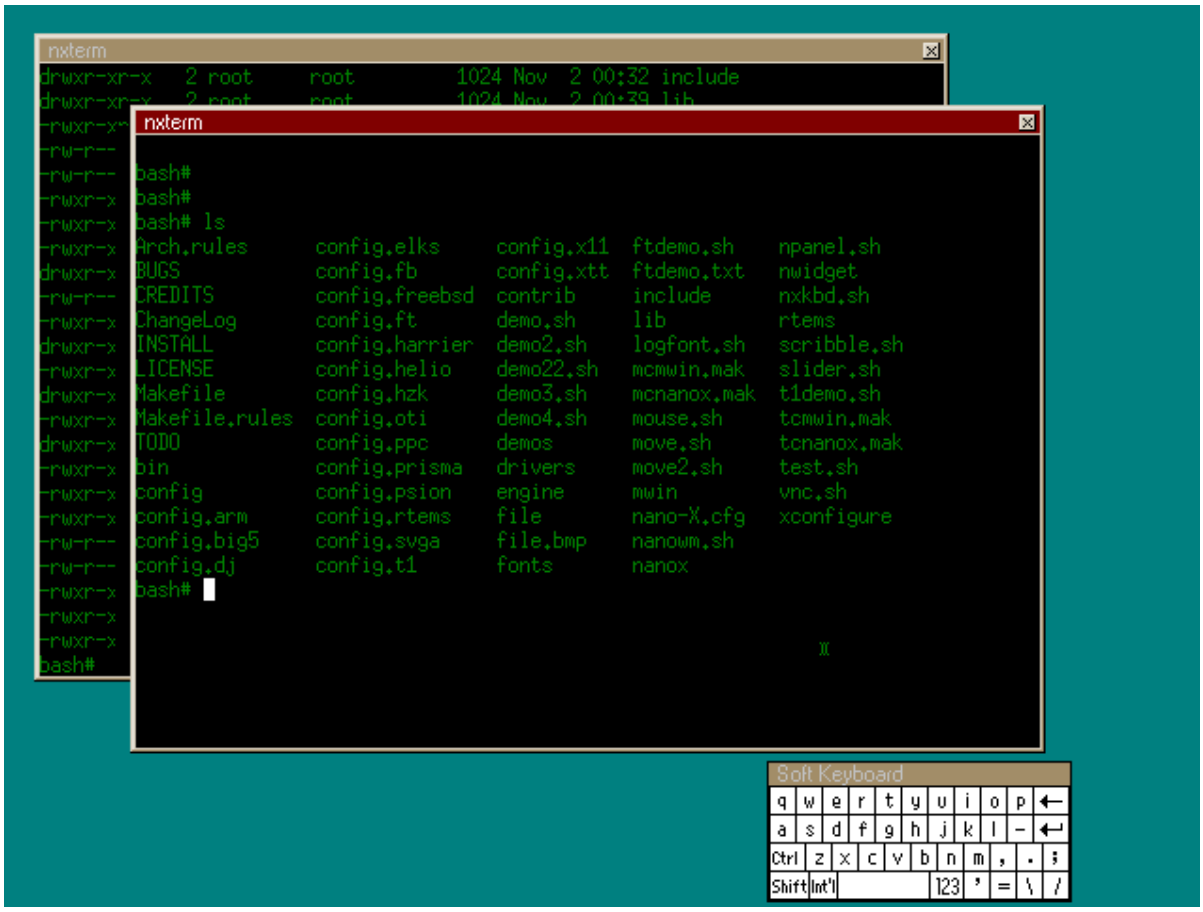
V záložce Device Drivers → HID Devices → Special HID drivers lze dále nalézt ovladače k většině běžných USB klávesnic a myší.

6.13.1 Okenní manažer Nano-X

Pro embedded systémy se nejčastěji používá systém Nano-X, který je také součástí distribuce. Nano-X je systém s vrstvenou strukturou, který umožňuje přepsat jednotlivé vrstvy a přizpůsobit systém potřebám konkrétní realizace [36].

Nejnižší vrstva umožňuje ovladačům obrazovky (klávesnice, myši a ostatního uživatelského vstupního hardware) přístup k aktuálnímu zobrazení.

Prostřední vrstva poskytuje grafické jádro, které umožňuje kreslení čar, polygonů, vyplňování objektů, atd. Obsahuje také barevné modely.
Poslední vrstva obsahuje různé procedury, funkce a třídy, které umožňují programátorům snadno vyvíjet grafické aplikace pro tento systém. Tato vrstva nemusí obsahovat pracovní plochu, ani vzhled oken.



Obr. 6.16 Nano-X systém, obrázek převzat z [36].

Pro zkompilování okenního manažeru Nano-X je potřeba v konfiguraci aplikací a knihoven zvolit následující volby:

Záložka MicroWindows

[*] MicroWindows

--- Compiling Options

[*] Optimize

[*] Verbose

[*] Thread Safe

--- Libraries

[*] Microwin

[*] NanoX

--- Applications

[*] NanoWM

--- Settings

[*] Have File IO

--- Display Config

[*] Frame Buffer Display
--- Mouse/Touch Screen
[*] Serial Mouse
--- Keyboard
[*] Scan keyboard
--- Install These Applications
[*] Nano-X
[*] NanoWM
[*] NXkbd
[*] NXterm

V případě potřeby lze přidat další demonstrační aplikace a knihovny v této záložce.
Systém Nano-X vyhodnocuje globální proměnnou s názvem CONSOLE, která udává, s jakou konzolí se bude pracovat. Proto je potřeba tuto proměnnou ještě před spuštěním okenního manažeru nastavit na jinou konzoli, než je ta, se kterou se aktuálně pracuje. Ke spuštění systému Nano-X lze použít následující sadu příkazů:

/>export CONSOLE=/dev/tty1
/>Nano-X &
/>nanowm &

Dále je již možno v tomto prostředí pracovat a spouštět aplikace, jako je virtuální terminál:

/>nxterm &

Jak již bylo uvedeno, umožňuje poslední vrstva systému Nano-X velice jednoduché psaní grafických aplikací. Aplikace pro Nano-X prostředí mohou být vytvářeny za použití „klient/server“ protokolu v síti, nebo soketu lokální domény, tzv. *local UNIX domain socket*. Jako velice jednoduchý příklad, který demonstruje vývoj grafických aplikací pro tento systém, může posloužit aplikace v příloze J. Tato aplikace vytvoří nové okno, v němž se následně pohybuje zleva doprava text „DEMO TEXT“.

6.13.2 Spuštění konzole v grafickém režimu

Kromě použití okenního manažeru Nano-X a psaní jednoduchých grafických aplikací lze Framebuffer využít také ke spuštění konzole operačního systému Linux v grafickém režimu. Pro tento režim zobrazení konzole je nutno v konfiguraci jádra přidat následující volby.

Záložka Device Drivers → Graphics support → Console display driver support

<*> Framebuffer Console support

[*] Select compiled-in fonts

[*] VGA 8x16 font

Záložka Device Drivers → Character devices

[*] Virtual terminal

[*] Enable character translations in console

[*] Support for console on virtual terminal

6.14 Obsluha přerušení, ovladač pro tlačítka

Obsluhu přerušení lze demonstrovat na příkladu použití tlačítek. Tato tlačítka jsou na desce celkem 4. Stejně jako v předchozích příkladech je i zde v nástroji SOPC builder, nebo Qsys nutno přidat další komponentu PIO (*Parallel I/O*). Je třeba provést toto nastavení komponenty: *width* (velikost): 4 bity, *direction* (směr): *Input ports only* (pouze vstupní porty) a v kartě *Input Options* je zaškrtnout volbu *Generate IRQ* (generovat požadavek na přerušení). Práce s přerušením v prostředí operačního systému Linux již byla popsána v kapitole 3.3. Ukázka ovladače pro tlačítka, která využívá přerušení je uvedena v příloze K. Ovladač po stisknutí tlačítka pošle zprávu do příkazové řádky systému a podle stisknutého tlačítka rozsvítí jednu ze zelených LED diod. Ve zdrojovém kódu je opět nutno nadefinovat adresy nastavené v programu SOPC Builder, nebo Qsys. V případě systému s podporou MMU je nutno adresy posunout o hodnotu 0xe0000000.

6.15 Nahrání systému do Flash paměti

Doposud bylo uvedeno pouze to, jak nakonfigurovat FPGA procesor, nahrát a spustit operační systém Linux z paměti SDRAM. To je výhodné pro testování systému, protože paměť Flash má omezený počet zápisu, který může být navíc pomalý. Výhodou však je, že Flash paměť umožní zkonfigurovat procesor a spustit distribuci operačního systému Linux při zapnutí

vývojové desky. Dle možností je možné využít CFI, nebo EPCS Flash paměť. Deska DE2-70 obsahuje paměťový čip EPCS16 s kapacitou 16 MB, který bude ve většině případů pro konkrétní aplikaci s distribucí uClinux postačovat. Následující postup popisuje, jak naprogramovat EPCS Flash paměť a použít zavaděč systému (*bootloader*) tak, aby došlo po zapnutí desky k automatickému nahrání systému z Flah paměti do SDRAM a k jeho následnému spuštění.

1. Nejprve je nutné přidat do systému v nástroji SOPC Builder, nebo Qsys novou komponentu EPCS Seriál Flash Controller.
2. Ve vlastnostech procesoru je nutné nastavit Reset Vector na tuto nově přidanou komponentu a Exception Vector na SDRAM, tak jak ukazuje následující obrázek.

Obr. 6.17 Nastavení vektoru pro reset a výjimku

3. Je třeba přeložit projekt a nakonfigurovat FPGA čip běžným způsobem:
   />nios2-configure-fpga /cesta/k/souboru.sof
4. Je třeba vytvořit soubor pro naprogramování paměti flash pro danou konfiguraci systému:
   />sof2flash --epcs –input=/cesta/k/souboru.sof –output=projekt.flash
5. Je třeba vytvořit soubor pro naprogramování paměti flash obsahující obraz distribuce operačního systému Linux a *boot loader*:
   />elf2flash --epcs --after=projekt.flash --output=linux.flash --boot=
   /cesta/boot_loader_epcs.srec
   Soubor **boot_loader_epcs.srec** lze nalézt ve složce, kam byl nainstalován software Quartus (např.: <C:\11.1\nios2eds\components\altera_nios2\boot_loader_epcs.srec>).
6. Je třeba naprogramovat Flash paměť:
   />nios2-flash-programmer --epcs --base=0x4002800 projekt.flash
   />nios2-flash-programmer --epcs --base=0x4002800 linux.flash
   V případě systému s MMU:
   />nios2-flash-programmer --epcs --mmu --base=0x4002800 projekt.flash
   />nios2-flash-programmer --epcs --mmu --base=0x4002800 linux.flash

`Base` je adresa epcs kontroléru, která byla nastavena v nástroji SOPC Builder nebo Qsys.

Po vypnutí a zapnutí desky dojde ke zkonfigurování FPGA obovdu a zavedení operační systém Linux do paměti SDRAM. Ve výchozím nastavení jádra čeká systém na připojení terminálu. Pokud má dojít k automatickému spuštění ještě pře připojením k zařízení, je potřeba v konfiguraci jádra vybrat následující volbu:

Záložka Device Drivers->Character devices->Serial drivers->

[*] Bypass output when no connection

7 Testy výkonu (Benchmark tesy)

Výkon systému byl testován pro všechny tři typy jader proceosru NIOS II. Byly také použity různé velikosti instrukční a datové cache paměti u jader, které danou paměť obsahují. Pro testování výkonu byly použity testy Dhrystone a Whetstone. Dále byla testována rychlost zápisu a čtení paměti SDRAM.

7.1 Parametry použitého procesoru NIOS II

Následující tabulka ukazuje konkrétní parametry jader použitého procesoru (EP2C70F896C6N) při frekvenci 100 MHZ.

Select a Nios II core:

	◉ Nios II/e	○ Nios II/s	○ Nios II/f
Nios II Selector Guide Family: Cyclone II f_{system}: 100,0 MHz cpuid: 0	**RISC** **32-bit**	RISC 32-bit **Instruction Cache** **Branch Prediction** **Hardware Multiply** **Hardware Divide**	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide **Barrel Shifter** **Data Cache** **Dynamic Branch Prediction**
Performance at 100,0 MHz	Up to 9 DMIPS	Up to 50 DMIPS	Up to 101 DMIPS
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache

Tab. 7.1 Přehled jader procesoru EP2C70F896C6N (konfigurace v nástroji SOPC Builder)

7.2 Testovací algoritmy

K měření výkonu byly použity dva testovací algoritmy. Algoritmus Dhrystone, který využívá k měření výkonu operace s celými čísly. Výpočty se podobají běžným instrukcím, které standardně procesor provádí. Tento algoritmus je průmyslovým standardem při měření a porovnávání výkonu procesorů. Druhým použitým algoritmem je Whetstone, který pro měření výkonu využívá plovoucí desetinou čárku. Výsledky těchto testů lépe odpovídají matematickému výpočetnímu výkonu procesorů. Kromě těchto testů byly provedeny také testy rychlosti zápisu a čtení z paměti SDRAM

7.2.1 Testovací algoritmus Dhrystone

První verzi tohoto algoritmu vytvořil v roce 1984 Dr. Reinhold P. Weicker. Algoritmus byl vytvořen za účelem měření výkonu počítačových systémů. Vzhledem k tomu, že již existoval algoritmus, který využíval plovoucí desetinnou čárku (tzv. Whetstone test), zaměřil se Weicker na testování výkonu pomocí výpočtů, které používají pouze celá čísla. Testovací algoritmus pracuje s aritmetickými a logickými operacemi a operacemi s řetězci. Současná verze 2.1 vytvořená v roce 1988 se používá dodnes a je považována za průmyslový standard v testování. Výsledkem testů je počet iterací za sekundu, nebo také jako číslo DMIPS (*Dhrystone Million Instructions executed Per Second*), neboli milion instrukcí vykonaných za vteřinu. Pro získání tohoto parametru je nutno vydělit počet iterací testu za sekundu čislem 1757. Tato hodnota je hodnotu, kterou v tomto testu získal první testovaný počítač (VAX11/780) [37].

7.2.2 Testovací algoritmus Whetstone

Tento test pro hodnocení výkonu počítačů napsal poprvé v jazyce FORTRAN Harold Curnow v roce 1972. Je využíván pro měření především matematického výkonu procesorů a využívá čísel s plovoucí desetinnou čárkou. Algoritmus používá jak jednoduché aritmetické operace s čísly s plovoucí desetinou čárkou, tak i výpočty s mocninami, logaritmy a trigonometrickými operacemi. Výsledkem testů je výkon procesoru, který je reprezentován jako WMIPS (*Whetstone Million Instructions executed Per Second*) a čas potřebný na provedení testu [38].

7.2.3 Testování rychlosti paměti

Rychlost paměti byla testována vytvořením, čtením a smazáním 10 MB souboru v paměti. K testu byly použity následující příkazy:
Vytvoření souboru: **/>time dd if=/dev/zero of=/file.dat bs=1M count=10**
Čtení souboru: **/>time dd if=/file.dat of=/dev/null**
Smazání souboru: />**time rm /file.dat**
Příkaz DD slouží k blokovému kopírování dat. První parametr udává blokové zařízení, ze kterého se data zkopírují. Druhý parametr udává jeho cílové umístění. Parametr bs určuje počet bajtů na blok a count počet bloků.

Příkaz TIME spustí předávaný příkaz a po jeho skončení vypíše dobu jeho běhu

Speciální zařízení </dev/null>, takzvané „Nulové“ se chová tak, že zahazuje všechna data, která jsou do něj zapsána. Jakýkoliv zápis končí úspěšně, čtení vrací stav „Dosažen konec souboru“ (EOF). Obdobně se chová speciální zařízení </dev/zero>, s tím rozdílem, že při čtení vrací tolik nulových znaků (binárních nul), kolik je požadováno. Používá se tak mimo jiné k vytváření prázdných souborů

7.3 Výsledky testů

Testy byly provedeny při frekvenci procesoru a paměti 100 MHz v distribuci uClinux s jádrem verze 3.3.0. Byla použita minimální konfigurace systému, která je potřebná pro běh operačního systému Linux na procesoru NIOS II. Byl testován jak systém s podporou MMU, tak i systém bez podpory MMU a to pro různé velikosti vyrovnávací paměti cache. Systém s podporou MMU byl testován pouze na jádře NIOS II/f, které má jako jediné možnost zapnout podporu jednotky správy paměti. Výkon systému bez podpory MMU byl testován na všech jádrech NIOS II/f, NIOS II/s a NIOS II/e. Testy byly provedeny pro různou konfiguraci cache pamětí u jader, které mají podporu této paměti, tzn. NIOS II/s a NIOS II/f. U jádra procesoru NIOS II/f byla konfigurována datová a instrukční cache paměť vždy na stejnou hodnotu. Jádro NIOS II/s obsahuje pouze instrukční cache paměť.

7.3.1 Výsledky testu Dhrystone

Pro tento test byl použit Dhrystone v 2.1 napsaný v jazyce C, který je součástí distribuce uClinux.

Výsledky testu při 1000000 iteracích jsou uvedeny v následujících tabulkách.

Systém bez podpory MMU, jádro NIOS II/e	
Doba trvání jednoho testu [µs]	42,4
Počet instrukcí vykonaných za 1s	23583,9
DMIPS	13,423
DMIPS/MHZ	0,134

Tab. 7.2 Výsledky Dhrystone testu, jádro NIOS II/e

Systém bez podpory MMU, jádro NIOS II/s							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Doba trvání jednoho testu [µs]	42,4	39,4	38,3	29	28,8	28,9	28,8
Počet instrukcí vykonaných za 1s	23584,4	25380,7	26109,7	34482,8	34722,2	34602,1	34722,2
DMIPS	13,423	14,445	14,860	19,626	19,762	19,694	19,762
DMIPS/MHZ	0,134	0,144	0,149	0,196	0,198	0,197	0,198

Tab. 7.3 Výsledky Dhrystone testu, jádro NIOS II/s

Systém bez podpory MMU, jádro NIOS II/f							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Doba trvání jednoho testu [µs]	34	29,2	27,1	8,4	8,4	5,9	5,2
Počet instrukcí vykonaných za 1s	29411,8	34246,6	36886,8	119047,0	119617,2	169204,7	191204,6
DMIPS	16,740	19,492	20,994	67,756	68,080	96,303	108,824
DMIPS/MHZ	0,167	0,195	0,210	0,678	0,681	0,963	1,088

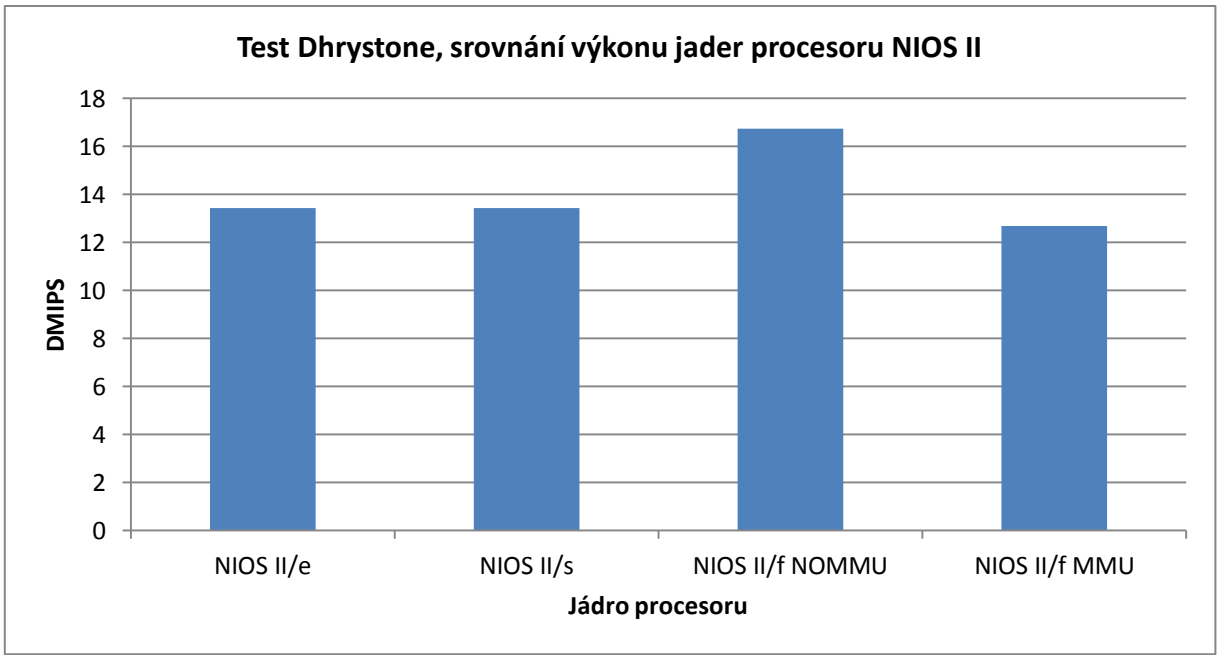
Tab. 7.4 Výsledky Dhrystone testu, jádro NIOS II/f, systém bez MMU

Systém s podporou MMU, jádro NIOS II/f							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Doba trvání jednoho testu [µs]	44,9	34,2	28,5	12,1	12,1	9,1	9,1
Počet instrukcí vykonaných za 1s	22271,7	29205,6	35087,7	82640	82850	110132,2	110497,2
DMIPS	12,676	16,622	19,970	47,035	47,154	62,682	62,890
DMIPS/MHZ	0,127	0,166	0,200	0,470	0,472	0,627	0,629

Tab. 7.5 Výsledky Dhrystone testu, jádro NIOS II/f, systém s MMU

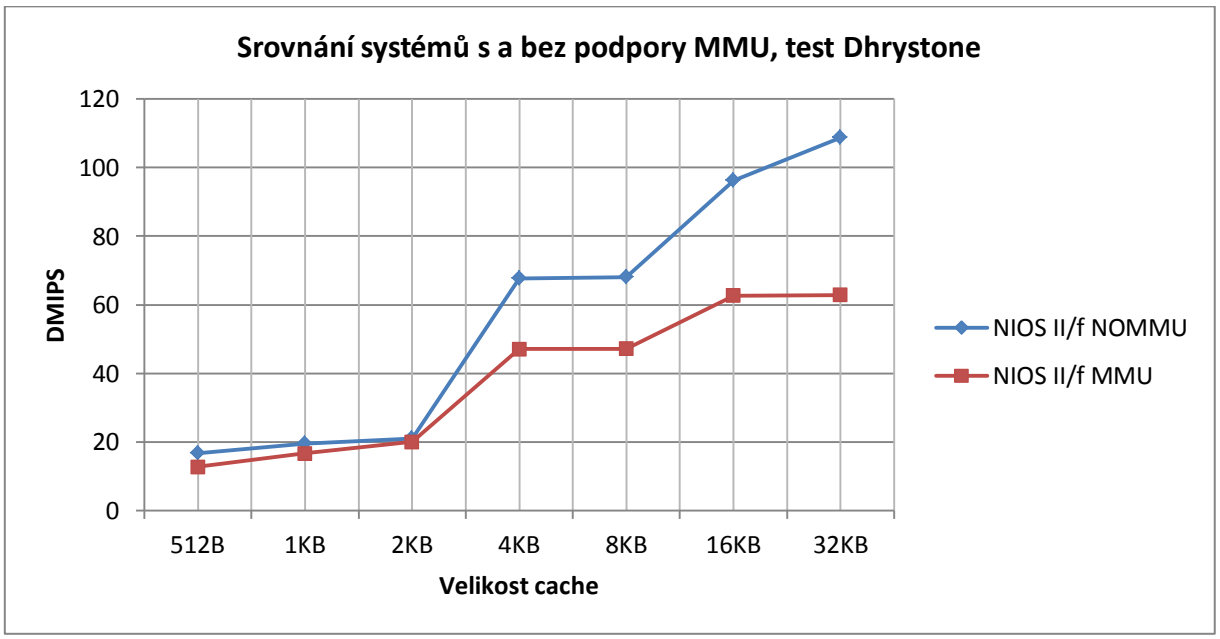
Graf 7.1 zobrazuje srovnání výsledků testu pro všechna tři jádra procesoru a také pro systém s podporou MMU u jádra NIOS II/f. Vzhledem k tomu, že jádro NIOS II/e neobsahuje žádnou paměť cache, byly pro srovnání použity výsledky s nastavenou minimální možnou hodnotou cache paměti u ostatních jader (512 B). Z grafu je patrné, že v tmto testu byl výkon

jádra NIOS II/e a NIOS II/s totožný. Jádro NIOS II/f dosáhlo lepších výsledků. Ve stejném čase zvládlo zpracovat o více než 6000 instrukcí algoritmu Dhrystone více. Dále je vidět pokles výkonu o téměř 20% při použití systému s podporou MMU.



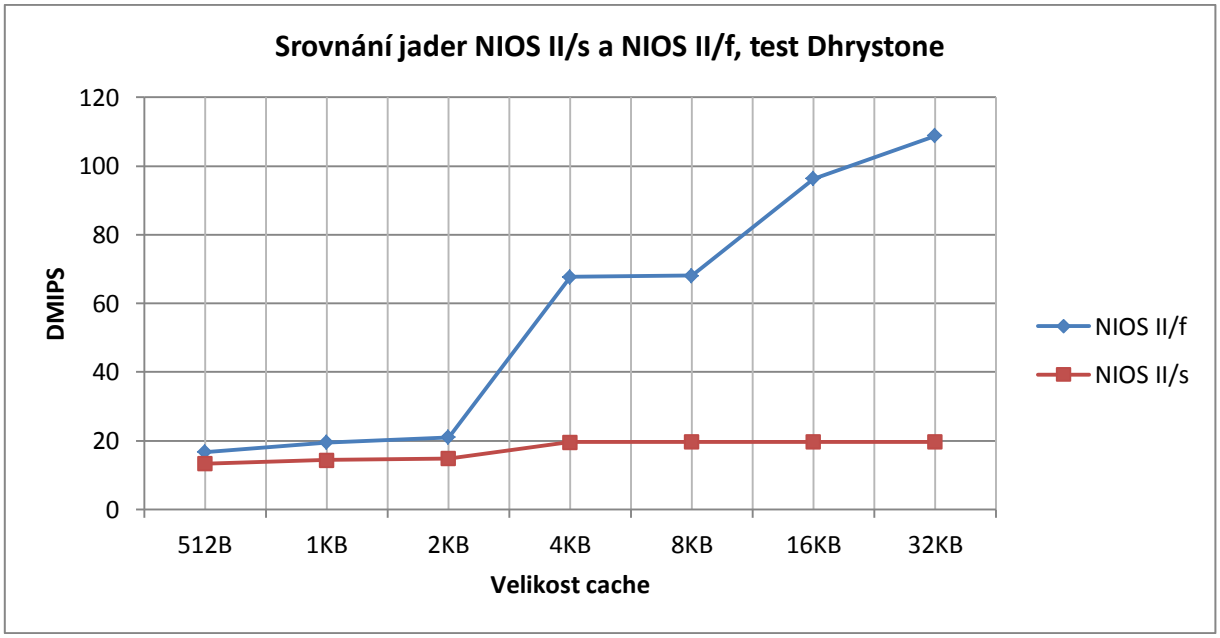
Graf 7.1 Srovnání výkonu jader, test Dhrystone

Graf 7.2 ukazuje srovnání operačního systému Linux s podporou MMU a bez podpory MMU na jádře procesoru NIOS II/f. Je vidět, že rozdíl ve výkonu narůstá se zvyšující se pamětí cache. Systém bez podpory MMU při 32 KB paměti cache nabízí téměř o 45% vyšší výkon při vykonávání operací s celými čísly, než systém s podporou MMU.



Graf 7.2 Srovnání výkonu systému s a bez podpory MMU, test Dhrystone

V grafu 7.3 je srovnán výkon jader NIOS II/s a NIOS II/f. V testu Dhrystone je vidět, že navyšování instrukční cache paměti u jádra NIOS II/s má na výsledný výkon v operačním systému Linux poměrně zanedbatelný vliv, což neplatí pro jádro NIOS II/f, kde je rozdíl mezi jádrem s 512 B a jádrem s 32 KB instrukční a datové vyrovnávací paměti cache velice markantní. Systém s 32 KB cache pamětí zvládl ve stejném čase vykonat téměř 65 krát více instrukcí algoritmu Dhrystone, než systém s 512 B cache. Toto číslo platí i ve srovnání s jádrem NIOS II/s.



Graf 7.3 Srovnání výkonu jádra NIOS II/s a NIOS II/f, test Dhrystone, systém bez MMU

7.3.2 Výsledky testu Whetstone

Opět byl použit testovací program obsažený v distribuci uClinux, který je z roku 1998 a je postaven na verzi získané z http://www.netlib.org/benchmark/whetstoned (verze z 16. května 1998).

Výsledky testů jsou uvedeny při 500 cyklech

Systém bez podpory MMU, jádro NIOS II/e	
Doba trvání testu [s]	186
WMIPS	0,268

Tab. 7.6 Výsledky Whetstone testu, jádro NIOS II/e

Systém bez podpory MMU, jádro NIOS II/s							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Doba trvání testu [s]	186	173	142	133	106	102	101
WMIPS	0,268	0,289	0,352	0,376	0,471	0,49	0,495

Tab. 7.7 Výsledky Whetstone testu, jádro NIOS II/s

Systém bez podpory MMU, jádro NIOS II/f							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Doba trvání testu [s]	100	89	64	58	40	38	37
WMIPS	0,5	0,561	0,781	0,862	1,3	1,3	1,4

Tab. 7.8 Výsledky Whetstone testu, jádro NIOS II/f, systém bez MMU

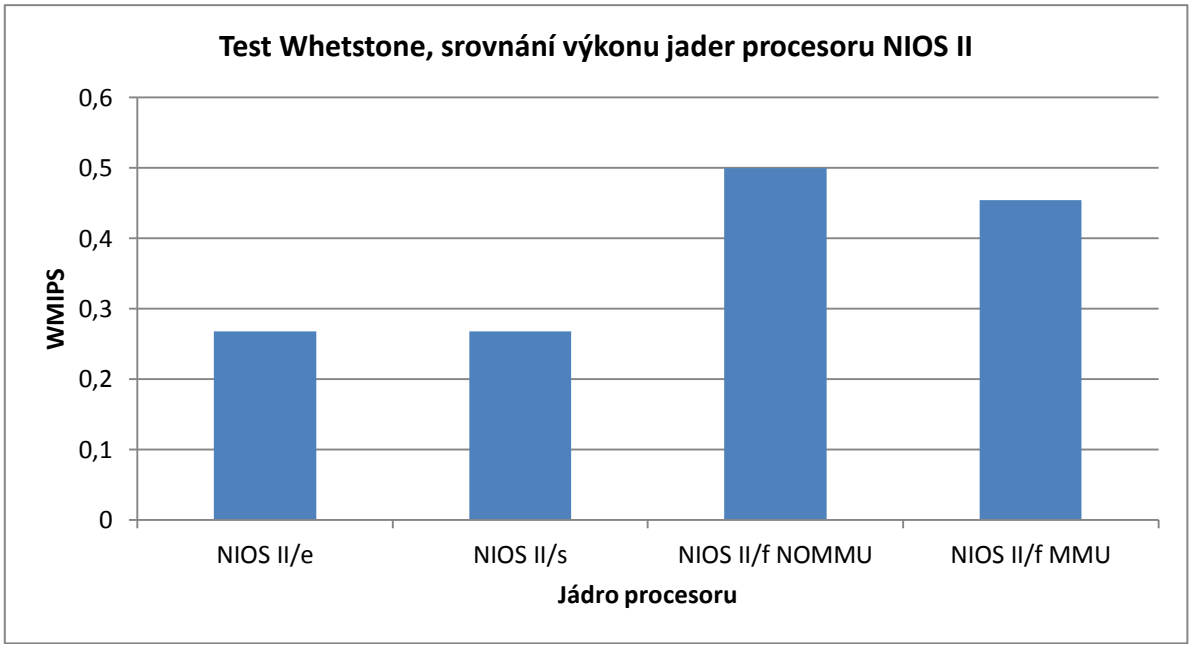
Systém s podporou MMU, jádro NIOS II/f							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Doba trvání testu [s]	110	97	81	56	47	42	39
WMIPS	0,454	0,515	0,617	0,892	1,1	1,2	1,3

Tab. 7.9 Výsledky Whetstone testu, jádro NIOS II/f, systém s MMU

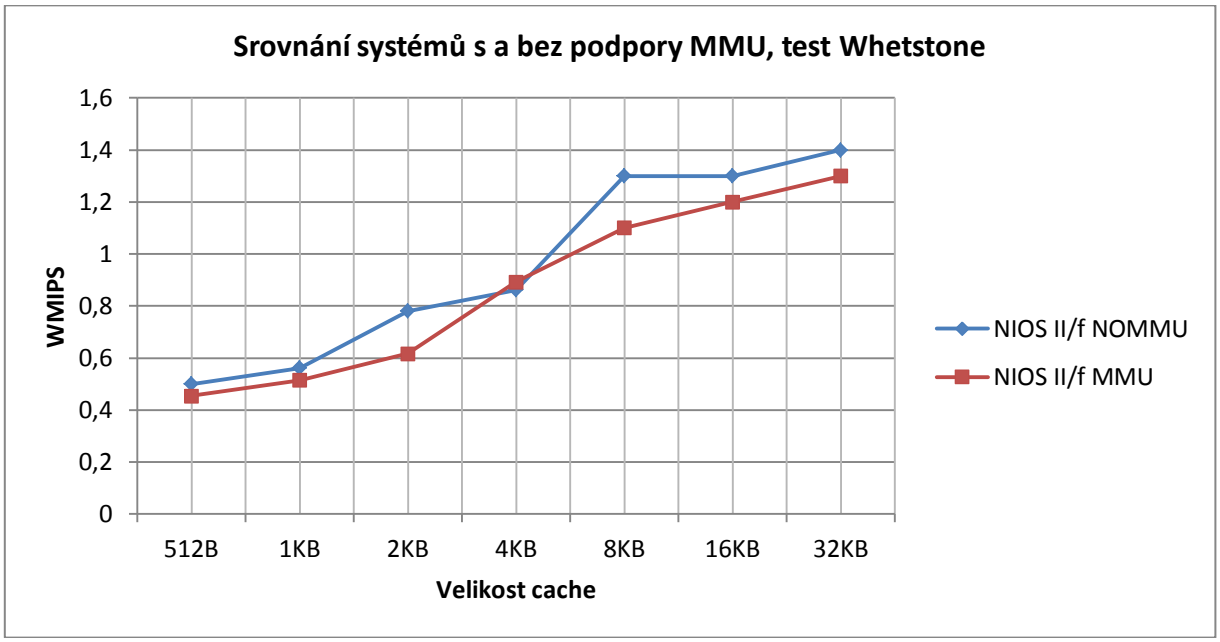
Následující graf 7.4 srovnává všechna tři jádra procesoru s minimální hodnotou cache paměti. Test s jádrem NIOS II/f je srovnáván se systém s podporou i bez podpory MMU. Podobně jako u systému Dhrystone je naměřený výkon jader NIOS II/e a NIOS II/s s 512 B cache paměti srovnatelný. Na jádře NIOS II/f byl výkon dosažený v tomto testu téměř dvojnásobný. U systému s podporou MMU je opět vidět pokles výkonu. Tneto pokles však není tak markantní jako při testování algoritmem Dhrystone.

Graf 7.5 opět ukazuje rozdíl mezi systémem s podporou MMU a bez této podpory. Podle předpokladů je vidět, že systém bez podpory MMU dosahuje lepších výsledků, než systém s podporou MMU. Tento rozdíl však není nikterak markantní.

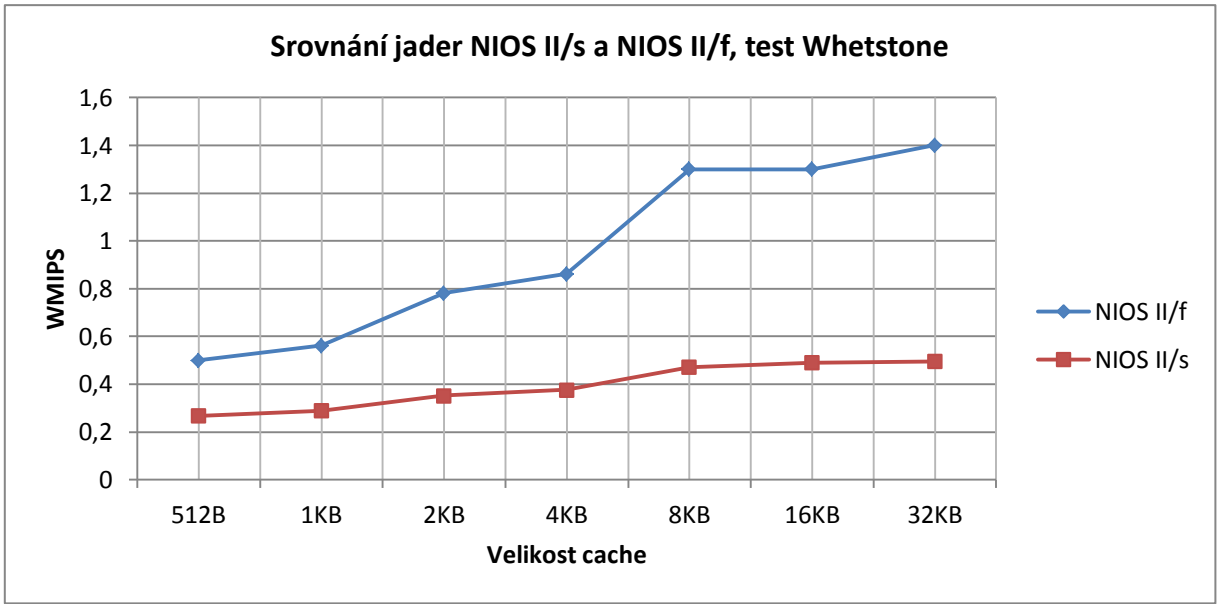
V grafu 7.6 je srovnáno jádro NIOS II/s s jádrem NIOS II/f v operačním systému Linux bez podpory MMU. Zde je vidět, že v případě jádra NIOS II/f je výkon znatelně ovlivněn velikostí použité paměti cache.



Graf 7.4 Srovnání výkonu jader, test Whetstone



Graf 7.5 Srování výkonu systému s a bez podpory MMU, test Whetstone



Graf 7.6 Srovnání výkonu jádra NIOS II/s a NIOS II/f, test Whetstone, systém bez MMU

7.3.3 Výsledky testování rychlosti paměti

Výsledky testování rychlosti práce s pamětí SDRAM jsou uvedeny v následujících tabulkách. Opět se jedná o různé konfigurace velikosti cache paměti.

Systém bez podpory MMU, jádro NIOS II/e	
Zápis 10MB dat [s]	2,13
Rychlost zápisu [MB/s]	4,69
Čtení 10MB dat [s]	5,32
Rychlost čtení [MB/s]	1,88
Mazání 10MB dat [s]	0,42
Rychlost mazání [MB/s]	23,81

Tab. 7.10 Test rychlosti SDRAM, jádro NIOS II/e

Systém bez podpory MMU, jádro NIOS II/s							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Zápis 10MB dat [s]	2,13	2,06	2,05	2	1,95	1,89	1,85
Rychlost zápisu [MB/s]	4,69	4,85	4,88	5,00	5,13	5,29	5,41
Čtení 10MB dat [s]	5,32	5,17	5,1	4,8	4,27	3,76	3,25
Rychlost čtení [MB/s]	1,88	1,93	1,96	2,08	2,34	2,66	3,08
Mazání 10MB dat [s]	0,42	0,36	0,33	0,29	0,28	0,28	0,28
Rychlost mazání [MB/s]	23,81	27,78	30,30	34,48	35,71	35,71	35,71

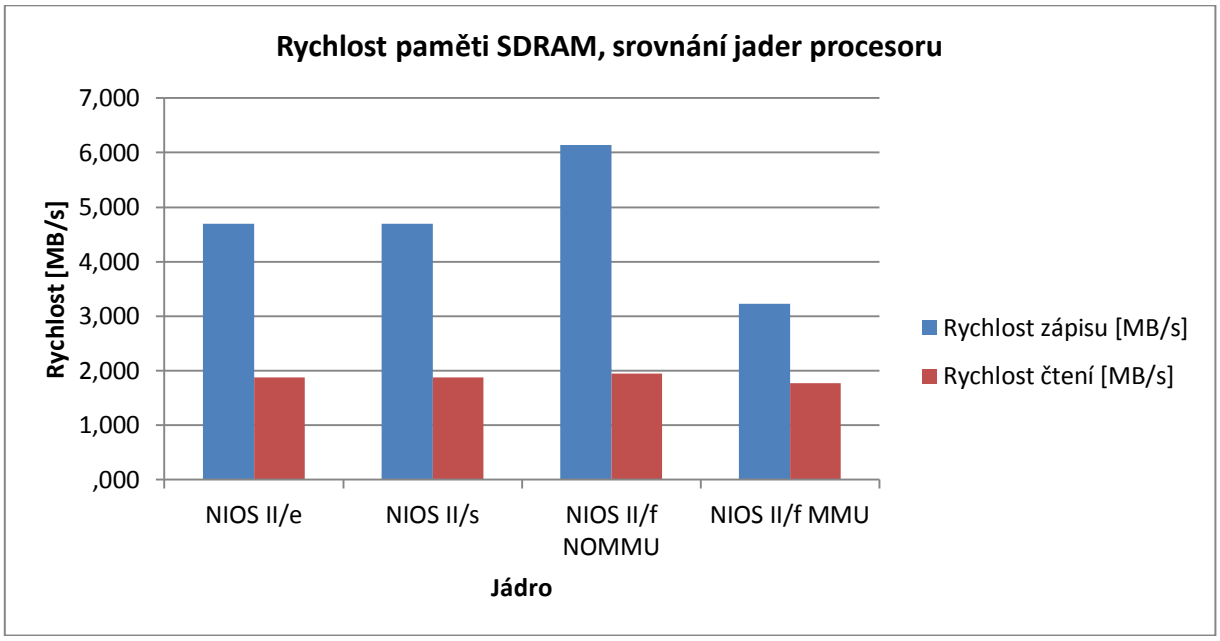
Tab. 7.11 Test rychlosti SDRAM, jádro NIOS II/s

Systém bez podpory MMU, jádro NIOS II/f							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Zápis 10MB dat [s]	1,63	1,53	1,49	1,42	1,3	1,18	1,14
Rychlost zápisu [MB/s]	6,13	6,54	6,71	7,04	7,69	8,47	8,77
Čtení 10MB dat [s]	5,14	4,26	3,42	2,82	1,61	1,19	0,81
Rychlost čtení [MB/s]	1,95	2,35	2,92	3,55	6,21	8,40	12,35
Mazání 10MB dat [s]	0,33	0,22	0,18	0,12	0,11	0,1	0,09
Rychlost mazání [MB/s]	30,30	45,45	55,56	83,33	90,91	100,00	111,11

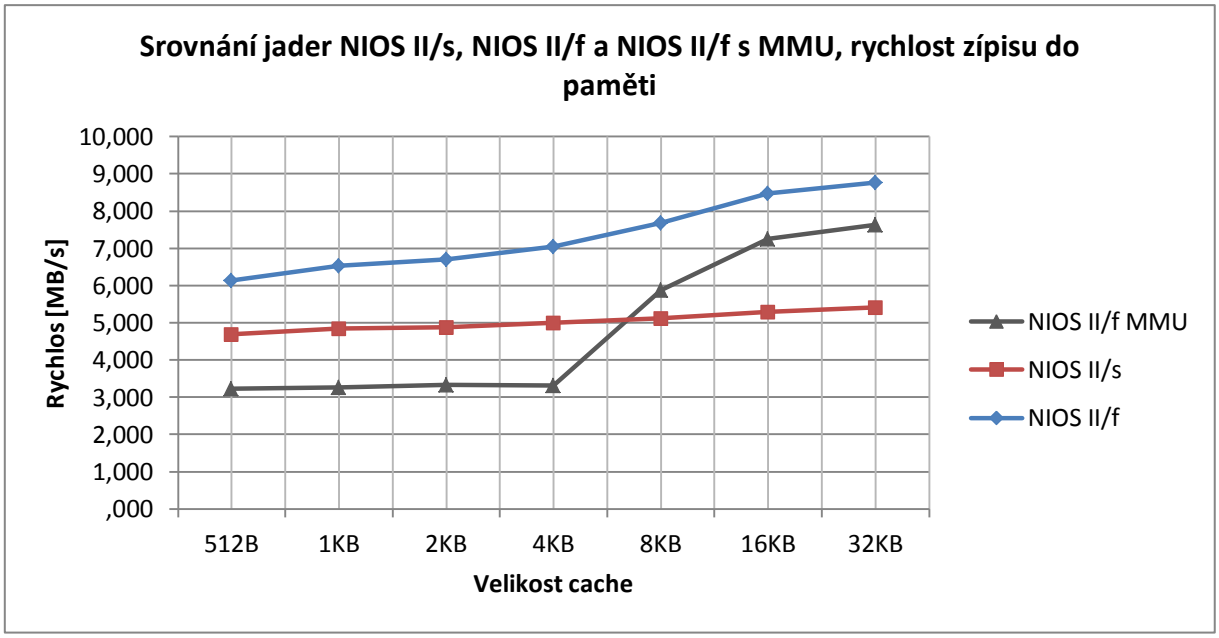
Tab. 7.12 Test rychlosti SDRAM, jádro NIOS II/f bez MMU

Systém s podporou MMU, jádro NIOS II/f							
Cache	512B	1KB	2KB	4KB	8KB	16KB	32KB
Zápis 10MB dat [s]	3,1	3,06	3	3,01	1,7	1,38	1,31
Rychlost zápisu [MB/s]	3,23	3,27	3,33	3,32	5,88	7,25	7,63
Čtení 10MB dat [s]	5,64	4,31	3,48	2,25	1,6	1,21	0,89
Rychlost čtení [MB/s]	1,77	2,32	2,87	4,44	6,25	8,26	11,24
Mazání 10MB dat [s]	0,28	0,22	0,15	0,13	0,11	0,1	0,09
Rychlost mazání [MB/s]	35,71	45,45	66,67	76,92	90,91	100,00	111,11

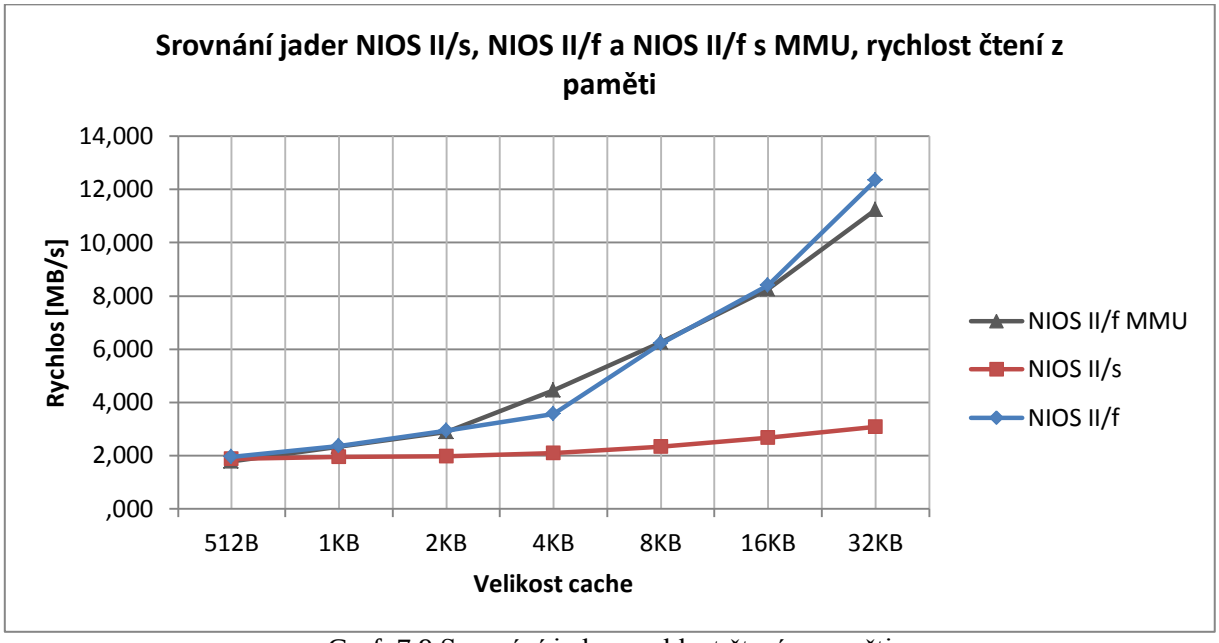
Tab. 7.13 Test rychlosti SDRAM, jádro NIOS II/f s MMU



Graf 7.7 Test rychlosti paměti SDRAM, srovnání jader



Graf 7.8 Srovnání jader, rychlost zápisu do pměti



Graf 7.9 Srovnání jader, rychlost čtení z paměti

7.4 Zhodnocení testů

Z tesů vyplívá, že jádro NIOS II/f s 32 KB cache pamětí dosahuje nejlepších výsledků ve všech testech. Výkon v testech u tohoto jádra je dále velmi závislý na velikosti chache paměti. Při srovnání systému s jádrem NIOS II/f, které obsahuje 512 B paměti cache a systému se stejnou konfigurací, který však obsahuje 32 KB paměti cache, je rozdíl v rychlosti těchto systémů více než dvojnásobný. Stejný rozdíl byl zaznamenán i v případě testu rychlosti zápisu

do paměti SDRAM. Pokud porovnáme jádra NIOS II/e a NIOS II/s, dosahují tato jádra v operačním systému Linux téměř stejných výsledků. U jádra NIOS II/s dochází vlivem použití větší paměti cache pouze k mírnému nárůstu výkonu. Proto je v případě rozhodování mezi těmito dvěma jádry nutno uvážit cenu licence, která je u jádra NIOS II/e zdarma. Cena roční licence pro ostatní jádra činní 495USD.

Pokud dojde ke srovnání systému s podporou MMU, kterou lze využít pouze u jádra NIOS II/f a systému se stejnou konfigurací bez podpory MMU. Dojde ke zjištění poklesu výkonu právě u systému s podporu MMU. Tento rozdíl je patrný zejména u testování algoritmem Dhrystone, kde se zvyšující se pamětí cache roste rozdíl mezi těmito dvěma systémy. Konkrétně tento rozdíl činil 45,934 DMIPS (téměř 45%) u systému s 32 KB cache paměti. U algoritmu Whetstone je rozdíl ve výkonu mezi systémem s podporou a bez podpory MMU pouze necelých 10%. To samé platí i pro rozdíl rychlost čtení z paměti SDRAM. U zápisu do paměti je rozdíl při použití méně než 8KB cache paměti téměř dvojnásobný.

Nejlepšího výsledku bylo dosaženo u jádra NIOS II/f s 32 KB cache pamětí a systému bez podpory MMU. V testu Dhrystone bylo pro tuto konfiguraci naměřeno 108,824 DMPIS a v testu Whetstone 1,4 WMIPS. Naopak nejhůře dopadly konfigurace s jádrem NIOS II/e a jádrem NIOS II/s s 512KB instrukční paměti cache. Oba tyto systémy získaly v testu Dhrystone 13,423 DMPIS a v testu Whetstone 0,268 WMIPS. Ještě horší výsledky v testu Dhrystone přinesla konfigurace s jádrem NIOS II/f s datovou a instrukční cache 512 B v systému s podporou MMU. Konkrétní naměřená hodnota u této konfigurace byla 12,676 DMIPS. V testu Whetstone pak tato konfigurace dosáhla hodnoty 0,454 WMIPS. U systému s podporou MMU a 32K cache pamětí bylo naměřeno 62,890 DMPIS a 1,3 WMIPS. Rychlost čtení z paměti SDRAM byla 11,24 MB a zápisu 7,63 MB.

Z grafů 7.8 a 7.9 je vidět, jak u jádra NIOS II/f vlivem větší paměti cache narůstá rychlost čtení a zápisu z a do paměti SDRAM.

Závěr

Na internetu lze najít několik společností, které nabízejí komerční distribuce operačního sytému Linux pro architekturu NIOS II. Výhodou těchto řešení zpravidla bývá otestovaný systém, včetně veškerého software, který je obsažen v distribucích. Další výhodou je, že společnosti ke svým distribucím poskytují v rámci licence profesionální podporu a servis. Nevýhodou těchto řešení zůstává jejich cena.

Dále je možné najít několik *open-source* řešení. Většinou se jedná o distribuce, které vycházejí z projektu uClinux a jsou upraveny pro konkrétní vývojové desky. V této oblasti se nejrychleji vyvíjí distribuce NIOS II Linux Community Edition. Tento projekt je spravován několika vývojáři, kteří se zabývají těmito procesory. Tato distribuce byla použita v praktické části diplomové práce. Během práce s touto distribucí se však ukázalo, že obsahuje několik chyb, zejména ve zdrojových kódech ovladačů. Vzhledem k tomu, že jsou veškerá problematika a vývoj distribuce probírány pouze v diskusních skupinách, je dalším podstatným problémem této distribuce chybějící dokumentace a popis existujících problémů. Veškeré informace lze nalézt pouze na internetových stránkách altera wiki [24], které však nejsou příliš aktuální a dále pak v diskusní skupině Nios2-dev [23] a fóru Altera [22]. Získání informací o této distribuci bylo poměrně složité a časově náročné. Vzhledem k této skutečnosti jsou v kapitole 6 uvedeny úpravy ovladačů pro periférie obsažené na desce a celkový postup implementace této distribuce na vývojový kit DE2-70 od spoečnosti Terasic.

V práci jsou diskutovány dvě verze operačního systému Linux. První verzí je systém bez podpory jednotky správy paměti (MMU). Výhodou tohoto systému je jednoduchý přístup k hardware z uživatelských aplikací a vyšší rychlost. Nevýhodou je především skutečnost, že uživatel může nevhodným zápisem do paměti způsobit pád celého systému. Druhou verzí je systém s podporou MMU, který tuto nevýhodu odstraňuje. To však přináší složitější vývoj uživatelských aplikací komunikujících s hardware a zpomalení systému. Je také nutný větší počet logických elementů FPGA obvodu. Vzhledem k tomu, že oba systémy mají uvedené výhody a nevýhody, jsou všechny postupy vždy uváděny pro obě verze těchto systémů.

Výsledkem této práce je kompletní distribuce operačního systému Linux pro vývojový kit DE2-70. Na přiloženém DVD lze najít jak zkompilovaný funkční obraz distribuce, tak její

zdrojové kódy. DVD obsahuje zkompilovaný obraz distribuce jak pro systém s podporou, tak bez podpory MMU. Přiložené DVD dále obsahuje soubory pro konfiguraci FPGA obvodu a referenční návrh systému v softwaru Quartus II, který byl použit. To umožňuje spustit distribuci na vývojovém kitu DE2-70 i bez hlukokých znalostí operačního systému Linux.

Modifikací zdrojových kódů lze distribuci přizpůsobit pro konkrétní aplikace a zařízení. V diplomové práci je dále na jednoduchých příkladech demonstrováno, jak z prostředí operačního systému Linux přistupovat k periferiím, které jsou obsaženy na vývojové desce DE2-70. V práci je detailně popsán postup vytvoření několika aplikací a ovladačů tak, aby je bylo snadné modifikovat pro konkrétní potřeby. Dále byl pro ukázku vytvořen CGI skript, který demonstruje možnosti přístupu k periferiím prostřednictvím webového rozhraní. Veškeré zdrojové kódy jsou obsaženy v příloze a rovněž dostupné na DVD, které je součástí této práce.

Vlastním přínosem této diplomové práce je zejména souhrn informací o možnostech nasazení operačního systému Linux na platformu NIOS II. Dalším přínosem jsou informace o odstranění problémů, které přináší distribuce NIOS II Community Edition. Diplomová práce zahrnuje ukázky uživatelských aplikací a ovladačů pro přístup k periferiím z prostředí operačního systém Linux pracujícím na architektuře NIOS II.

Použitá literatura

[1] ALTERA. *Nios II Processor Reference Handbook* [online]. NII5V1-11.0. San Jose, CA 95134, 2011, May 2011 [cit. 2012-02-26]. Dostupné z: http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

[2] Nios II Processor. ALTERA. *FPGA CPLD and ASIC from Altera* [online]. [cit. 2012-02-27]. Dostupné z: http://www.altera.com/devices/processor/nios2/ni2-index.html

[3] NIOS II Performance Becnchmarks. ALTERA. *FPGA CPLD and ASIC from Altera* [online]. 2011, 5 Jun 2011 [cit. 2012-05-08]. Dostupné z: http://www.altera.com/literature/ds/ds_nios2_perf.pdf

[4] TORVALDS, Linus. *The Linux Kernel Archives* [online]. 2012 [cit. 2012-03-21]. Dostupné z: http://kernel.org

[5] Embedded Linux kernel usage. FREE ELECTRONS. *Free Electrons*: *Embedded Linux Experts* [online]. © Copyright 2004-2009, Feb 21, 2011 [cit. 2012-05-08]. Dostupné z: http://free-electrons.com/doc/embedded-linux-kernel-usage.pdf

[6] Toolchains. *Embedded Linux Wiki* [online]. 2011, last modified on 27 October 2011, at 12:17 [cit. 2012-03-18]. Dostupné z: http://elinux.org/Toolchains

[7] ANDERSEN, Erik. *BusyBox* [online]. 22 April 2012, © 1999-2008 [cit. 2012-05-07]. Dostupné z: http://www.busybox.net/

[8] Gentoo Linux Documentation: Bash by example, Part 3. GENTOO FOUNDATION, Inc. *Gentoo Linux* [online]. 2005, October 9, 2005 [cit. 2012-05-08]. Dostupné z: http://www.gentoo.org/doc/en/articles/bash-by-example-p3.xml

[9] Using the GNU Compiler Collection (GCC). FREE SOFTWARE FOUNDATION, Inc. *GCC, the GNU Compiler Collection* [online]. © 1988, 1989, 1992, 1993, 1994, 1995, 1996, 1997, [cit. 2012-05-08]. Dostupné z: http://gcc.gnu.org/onlinedocs/gcc-4.7.0/gcc/

[10] STONES, Richard. *Linux začínáme programovat*. 1. vyd. Praha: Computer Press, 2000, 897 s. ISBN 80-722-6307-2.

[11] MITCHELL, Mark, Jeffrey OLDHAM a Alex SAMUEL. *Advanced Linux programming* [online]. 1st ed. Indianapolis, Ind.: New Riders Pub., 2001, 340 s. [cit. 2012-05-08]. ISBN 07-357-1043-0. Dostupné z: http://i.iinfo.cz/files/root/k/ALP.pdf

[12] CORBET, Jonathan. *Linux device drivers* [online]. 3rd ed. Sebastopol: O´Reilly, 2005, 615 s. [cit. 2012-05-08]. ISBN 05-960-0590-3. Dostupné z: http://lwn.net/Kernel/LDD3/

[13] SEDLÁČEK, Daniel. Píšeme operační systém: přerušení. *Root.cz* [online]. 20. 2. 2006 [cit. 2012-05-08]. ISSN 1212-8309. Dostupné z: http://www.root.cz/clanky/piseme-operacni-system-preruseni/

[14] The Linux Kernel Module Programming Guide: Chapter 12. Interrupt Handlers. SALZMAN, Peter Jay. *The Linux Documentation Project* [online]. ver 2.4.0. © 2001, 2003-04-04 [cit. 2012-05-08]. Dostupné z: http://tldp.org/LDP/lkmpg/2.4/html/x1210.html

[15] DIONNE, D. Jeff a Michael DURRANT. ARCTURUS NETWORKS INC. *UClinux*: *Embedded Linux/Microcontroller Project* [online]. © 1998 - 2002 D. Jeff Dionne and Michael Durrant C, April 1, 2012 [cit. 2012-05-08]. Dostupné z: http://www.uclinux.org/

[16] NIKKANEN, Kimmo. *UCLINUX AS AN EMBEDDED SOLUTION* [online]. Turku, Salo, Finland, 2003 [cit. 2012-05-08]. Dostupné z: http://koti.mbnet.fi/knikkane/uclin_release1.pdf. Bachelor's Thesis. Turku Polytechnic. Vedoucí práce Jari-Pekka Paalassalo, Lic.Sc.

[17] Position Independent Code (PIC) in shared libraries. BENDERSKY, Eli. *Eli Bendersky's website* [online]. 2011, November 3rd, 2011 at 6:14 am [cit. 2012-05-08]. Dostupné z: http://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-libraries/

[18] Wind River Linux 4. WIND RIVER. *Wind River* [online]. Rev. 08/2011. 2011, 08/2011 [cit. 2012-05-08]. Dostupné z: http://www.windriver.com/products/product-notes/PN_Linux_4_1_0811.pdf

[19] TIMESYS. *Embedded Linux From A Trusted Source | Timesys Embedded Linux* [online]. 2012 [cit. 2012-05-08]. Dostupné z: http://timesys.com/

[20] Timesys Factory(TM) — 'No Compromise' Custom Platform Builder. TIMESYS CORPORATION. *Embedded Linux From A Trusted Source | Timesys Embedded Linux* [online]. 2012

[21] Board Support Packages. SYSTEM LEVEL SOLUTIONS. *SLSCORP* [online]. © 2012 [cit. 2012-05-08]. Dostupné z: http://www.slscorp.com/downloads/board-support-pkgs/

[22] Linux Forum. *Altera Forum* [online]. 2007-2012 [cit. 2012-05-08]. Dostupné z: http://www.alteraforum.com/forum/forumdisplay.php?f=50

[23] Nios2-dev Info Page. *SOPC git server* [online]. National Taiwan University of Science and Technology [cit. 2012-05-08]. Dostupné z: http://sopc.et.ntust.edu.tw/cgi-bin/mailman/listinfo/nios2-dev

[24] *Altera Wiki* [online]. 2010-2012, 5 August 2011, at 16:29 [cit. 2012-05-08]. Dostupné z: http://www.alterawiki.com/

[25] Altera DE2-70: Layout. TERASIC. *Terasic Technologies* [online]. [cit. 2012-05-07]. Dostupné z: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=53&No=226&PartNo=3

[26] Downloading Linux Distribution. *Altera Wiki* [online]. 2010-2011, 12 July 2011, at 14:42 [cit. 2012-05-09]. Dostupné z: http://www.alterawiki.com/wiki/Downloading_Linux_Distribution

[27] KROAH-HARTMAN, Greg. The Kernel Configuration and Build Process. *Linux Journal*: *The Original Magazine of Linux Community* [online]. May 01, 2003 [cit. 2012-05-09]. Dostupné z: http://www.linuxjournal.com/article/6568

[28] SOBELL, Mark G. *Linux*: *praktický průvodce*. 1. vyd. Praha: Computer Press, 1999, 946 s. ISBN 80-722-6190-8.

[29] *FDTWiki* [online]. 28 October 2010, at 06:11 [cit. 2012-05-09]. Dostupné z: http://devicetree.org

[30] Linux / Unix Command: inittab. THE NEW YORK TIMES COMPANY. *About.com*: *Linux* [online]. © 2012 [cit. 2012-05-09]. Dostupné z: http://linux.about.com/od/commands/l/blcmdl5_inittab.htm

[31] BARR, Tavis, Nicolai LANGFELDT, Seth VIDAL a Tom MCNEAL. *Linux NFS−HOWTO* [online]. Revision v3.1. 2002−08−25 [cit. 2012-05-09]. Dostupné z: http://tldp.org/HOWTO/pdf/NFS-HOWTO.pdf

[32] KUHLMANN, Gero a Martin MARES. Mounting the root filesystem via NFS (nfsroot). *The Linux Kernel Archives* [online]. 1996, 1997 [cit. 2012-05-09]. Dostupné z: http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/nfsroot.txt

[33] JANÍK, Pavel. Webový server thttpd. *LINUXZONE* [online]. 04. 02. 2002, 00:01 [cit. 2012-05-09]. Dostupné z: http://www.linuxzone.cz/index.phtml?ids=6&idc=53

[34] JOHNSTON, Matt. Dropbear SSH server and client. *Matt Johnston* [online]. 24 February 2012 [cit. 2012-05-09]. Dostupné z: https://matt.ucc.asn.au/dropbear/dropbear.html

[35] Framebuffer. *Ubuntu Wiki* [online]. 2010-09-30 04:57:27 [cit. 2012-05-09]. Dostupné z: https://wiki.ubuntu.com/FrameBuffer

[36] *The Nano-X Window System* [online]. 2010 [cit. 2012-05-08]. Dostupné z: http://www.microwindows.org

[37] YORK, Richard. ARM LTD. *Benchmarking in context: Dhrystone* [online]. March 2002 [cit. 2012-05-09]. Dostupné z: http://www.iuma.ulpgc.es/~nunez/procesadoresILP/DhrystoneMIPS-CriticismbyARM.pdf

[38] Whetstone (benchmark). In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2011, 4 May 2011 at 21:01. [cit. 2012-05-09]. Dostupné z: http://en.wikipedia.org/wiki/Whetstone_(benchmark)

[39] KRAUS, Václav. *Porovnání syntetizovatelných softwarových procesorových jader*. Plzeň, 2009. Diplomová práce. Západočeská univerzita v Plzni. Fakulta elektrotechnická. Katedra aplikované elektrotechniky a telekomunikací. Vedoucí práce Ing. Martin Poupa, Ph.D.

[40] HEROUT, Pavel. *Učebnice jazyka C*. Vyd. 1. České Budějovice: Kopp, 2001, 269 s. ISBN 80-858-2821-9.

Přílohy

Příloha A – Konfigurace ovladače pro síťový adaptér DM9000 (soubor linux-2.6/arch/nios2/kernel/setup.c)

```
#define DM9000A_BASE 0x4001028
#define DM9000A_IRQ 2
#include <linux/dm9000.h>
#include <linux/irq.h>
#include <linux/device.h>
#include <linux/interrupt.h>
#include <linux/platform_device.h>
static struct resource dm9k_resource[] = {
[0] = {
       .start = DM9000A_BASE,
       .end = DM9000A_BASE + 3,
       .flags = IORESOURCE_MEM,
},
[1] = {
       .start = DM9000A_BASE + 4,
       .end = DM9000A_BASE + 4 + 3,
       .flags = IORESOURCE_MEM,
       },
[2] = {
       .start = DM9000A_IRQ,
       .end = DM9000A_IRQ,
       .flags = IORESOURCE_IRQ | IRQF_TRIGGER_HIGH,
       }
};
static struct dm9000_plat_data dm9k_platdata = {
       .flags = DM9000_PLATF_16BITONLY,
   .dev_addr = {01,01,01,01,01,01},
};
static struct platform_device dm9k_device = {
       .name = "dm9000",
       .id = 0,
       .num_resources = ARRAY_SIZE(dm9k_resource),
       .resource = dm9k_resource,
       .dev = {
       .platform_data = &dm9k_platdata,
       }
};
static int __init dm9k_device_init(void)
{
       /* customizes platform devices, or adds new ones */
       platform_device_register(&dm9k_device);
       return 0;
}
arch_initcall(dm9k_device_init);
```

Příloha B – správný konfigurační soubor pro ovladač DM9000 (linux-2.6/drivers/net/ethernet/davicom/Kconfig)

```
     config DM9000
            tristate "DM9000 support"
            depends on ARM || BLACKFIN || MIPS || NIOS2
            select CRC32
            select NET_CORE
            select MII
            ---help---
              Support for DM9000 chipset.

              To compile this driver as a module, choose M here.  The module
              will be called dm9000.
```

Příloha C – úprava ovladače LCD 16207 pro systém s podporou MMU (soubor lcd_16207.c)

```
--- /home/fandar/mmu/lcd16207-kernel/lcd_16207.c      2007-11-04 13:36:02.000000000
+0100
+++ ../linux-2.6/drivers/char/lcd_16207.c      2012-04-13 21:47:07.592869317 +0200
@@ -9,8 +9,12 @@
 #include <linux/fs.h>
 #include <asm/uaccess.h>  /* for get_user and put_user */
 #include <linux/delay.h>
-
+#include <linux/mman.h>
+#include <linux/fcntl.h>
+#include <linux/unistd.h>
 #include "lcd_16207.h"
+#include <linux/cdev.h>
+#include <linux/init.h>

 #define SUCCESS 0
 //#define DEBUG 1
@@ -46,11 +50,40 @@
 static char *Message_Ptr_Line2 = Message + BUF_LCD_LINE;
 static char *Message_Ptr_Write;

+static struct device_dev_t
+{
+  int __iomem*  membase;
+  struct cdev   cdev;
+} s_device_dev;
+
+
+static int device_mmap(struct file *filp, struct vm_area_struct * vma)
+{
+  struct device_dev_t *dev = (struct device_dev_t*)filp->private_data;
+
+  vma->vm_flags |= VM_MAYSHARE | VM_SHARED;
+
+  vma->vm_start = (unsigned long)dev->membase;
+  vma->vm_end = vma->vm_start + 8;
+
+  return 0;
+}
+
+static unsigned long device_get_unmapped_area(struct file *file,
+                                      unsigned long addr,
+                                      unsigned long len,
+                                      unsigned long pgoff,
+                                      unsigned long flags)
+{
+  return pgoff << PAGE_SHIFT;
+}
+
 /*
  * This is called whenever a process attempts to open the device file
  */
 static int device_open(struct inode *inode, struct file *file)
 {
+struct device_dev_t *dev = container_of(inode->i_cdev, struct device_dev_t, cdev);
 #ifdef DEBUG
   printk(KERN_INFO "device_open(%p)\n", file);
 #endif
@@ -200,8 +233,7 @@
  * calling process), the ioctl call returns the output of this function.
  *
  */
-static int device_ioctl(struct inode *inode,   /* see include/linux/fs.h */
-             struct file *file,  /* ditto */
```

```
+static long device_ioctl( struct file *file,   /* ditto */
               unsigned int ioctl_num,    /* number and param for ioctl */
               unsigned long ioctl_param)
 {
@@ -279,11 +311,13 @@
 struct file_operations Fops = {
   .read = device_read,
   .write = device_write,
-  .ioctl = device_ioctl,
+  .unlocked_ioctl = device_ioctl,
   .open = device_open,
   .release = device_release,     /* a.k.a. close */
+  .mmap    = device_mmap,
 };

+
 /*
  * Initialize the module - Register the character device
  */
@@ -293,6 +327,8 @@
   /*
    * Register the character device (atleast try)
    */
+  struct device_dev_t *dev = &s_device_dev;
+  dev->membase = ioremap(na_lcd_16207, 8);
   ret_val = register_chrdev(MAJOR_NUM, DEVICE_NAME, &Fops);

   /*
@@ -359,13 +395,13 @@
   /*
    * Unregister the device
    */
-  ret = unregister_chrdev(MAJOR_NUM, DEVICE_NAME);
+  unregister_chrdev(MAJOR_NUM, DEVICE_NAME);

   /*
    * If there's an error, report it
-   */
+
   if (ret < 0)
-    printk(KERN_ALERT "Error: unregister_chrdev: %d\n", ret);
+    printk(KERN_ALERT "Error: unregister_chrdev: %d\n", ret); */
 }


@@ -416,7 +452,13 @@
 //write 32bit value to avalon bus address
 static void WriteNios(unsigned long addr, unsigned long value)
 {
-  (* (volatile unsigned long *)(addr))=value;
+struct device_dev_t *dev = &s_device_dev;
+volatile unsigned int *addrf;
+  cdev_init(&dev->cdev, &Fops);
+  dev->cdev.owner = THIS_MODULE;
+  dev->membase = ioremap(na_lcd_16207, 8);
+  outb(value, addr);
+(* (volatile unsigned long *)(addr))=value;
 }

 //read 32bit value from avalon bus address
@@ -443,13 +485,3 @@

 module_init(init_module);
 module_exit(cleanup_module);
-
-
-#define author "The kernel programming guide, NIOS wiki users and mamba"
```

Příloha D – úprava ovladače LCD 16207 pro systém s podporou MMU (soubor lcd_16207.h)

```
--- /home/fandar/mmu/lcd16207-kernel/lcd_16207.h      2007-11-04 13:36:10.000000000
+0100
+++ ../linux-2.6/drivers/char/lcd_16207.h      2012-04-13 19:46:04.696176223 +0200
@@ -18,6 +18,8 @@
   */
 #define MAJOR_NUM 250

+#define na_lcd_16207 0x00002440 //BASE ADDRESS
+#define na_lcd_16207_0 na_lcd_16207 | 0xe0000000
 #define ADR_LCD_COMMAND na_lcd_16207_0
 #define ADR_LCD_READY (na_lcd_16207_0 +4)
 #define ADR_LCD_DATA (na_lcd_16207_0 + 8)
```

Příloha E – úprava ovladače LCD 16207 pro systém bez podpory MMU (soubor lcd_16207.c)

```
--- lcd_16207.c     2012-05-05 21:47:36.465452978 +0200
+++ lcd_nommu_new.c 2012-04-13 12:24:49.781011039 +0200
@@ -200,8 +200,7 @@
   * calling process), the ioctl call returns the output of this function.
   *
   */
-static int device_ioctl(struct inode *inode,   /* see include/linux/fs.h */
-              struct file *file,  /* ditto */
+static long device_ioctl( struct file *file,   /* ditto */
               unsigned int ioctl_num,   /* number and param for ioctl */
               unsigned long ioctl_param)
 {
@@ -279,7 +278,7 @@
 struct file_operations Fops = {
   .read = device_read,
   .write = device_write,
-  .ioctl = device_ioctl,
+  .unlocked_ioctl = device_ioctl,
   .open = device_open,
   .release = device_release,     /* a.k.a. close */
 };
@@ -359,13 +358,13 @@
   /*
    * Unregister the device
    */
-  ret = unregister_chrdev(MAJOR_NUM, DEVICE_NAME);
+  unregister_chrdev(MAJOR_NUM, DEVICE_NAME);

   /*
    * If there's an error, report it
-   */
+
   if (ret < 0)
-    printk(KERN_ALERT "Error: unregister_chrdev: %d\n", ret);
+    printk(KERN_ALERT "Error: unregister_chrdev: %d\n", ret); */
 }


@@ -443,7 +442,3 @@

 module_init(init_module);
 module_exit(cleanup_module);
-
-
-
-
```

Příloha F – úprava ovladače LCD 16207 pro systém bez podpory MMU (soubor lcd_16207.h)

```
--- lcd_16207_old.h 2007-11-04 12:36:10.000000000 +0100
+++ lcd_16207.h     2012-05-05 21:53:46.000000000 +0200
@@ -18,10 +18,11 @@
   */
 #define MAJOR_NUM 250

-#define ADR_LCD_COMMAND na_lcd_16207_0
-#define ADR_LCD_READY (na_lcd_16207_0 +4)
-#define ADR_LCD_DATA (na_lcd_16207_0 + 8)
-#define ADR_LCD_READ (na_lcd_16207_0 + 12)
+#define na_lcd_16207 0x00002440 #SOPC base address
+#define ADR_LCD_COMMAND na_lcd_16207_0 + 0x80000000
+#define ADR_LCD_READY (na_lcd_16207_0 + 0x80000000 + 4)
+#define ADR_LCD_DATA (na_lcd_16207_0 + 0x80000000 + 8)
+#define ADR_LCD_READ (na_lcd_16207_0 + 0x80000000 + 12)

 #define ADR_LCD_LINE1 0x80 + 0x00
 #define ADR_LCD_LINE2 0x80 + 0x40
```

Příloha G – CGI skript pro ovlání LED diod, LCD displeje a zobrazení stavu přepínačů

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
//BASE ADDRESS
#define GPIO_BASE 0x00002420 //Zelene LED
#define GPIO_BASE_R 0x00002480 //Cervene LED
#define GPIO_BASE_SW 0x00002470 //Prepinace
int main ()
{
  char *data;
  int statuslcd,statusr,statusg,statussw,i;
  char m[100];
  unsigned short l,s;
  unsigned int r,ii;

  printf("%s%c%c\n",
"Content-Type:text/html;charset=iso-8859-1",13,10); //typ dokumentu

  ///LCD 16207 HTML kod///
  printf("<TABLE><TR><TD width='300px'>");
  printf("<TITLE>DE2_70 Control</TITLE>\n");
  printf("<H3>WRITE TEXT TO LCD 16207</H3>\n");
  data = getenv("QUERY_STRING");
  statuslcd=sscanf(data,"m=%s",m); //precetni odeslane hodnoty
  printf ("<FORM><INPUT name='m'><input type='submit' value='SET'></FORM>\n");

  //Zelene LED HTML kod///
  printf("<H3>TURN ON GREEN LEDs</H3>\n");
  printf  ("<script language='javascript'>var gall=0; function cong(i,act)
{i=parseInt(i); if (act==1) gall=gall+i; else gall=gall-i;
document.getElementById('l').value=gall.toString(16);}</script>");
  printf ("<LABEL>LED N. 0 </LABEL><INPUT type='checkbox' name='r' value='1'
onclick='cong(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 1 </LABEL><INPUT type='checkbox' name='r' value='2'
onclick='cong(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 2 </LABEL><INPUT type='checkbox' name='r' value='4'
onclick='cong(this.value,this.checked);' ><BR>\n");
```

```
  printf ("<LABEL>LED N. 3 </LABEL><INPUT type='checkbox' name='r' value='8'
onclick='cong(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 4 </LABEL><INPUT type='checkbox' name='r' value='16'
onclick='cong(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 5 </LABEL><INPUT type='checkbox' name='r' value='32'
onclick='cong(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 6 </LABEL><INPUT type='checkbox' name='r' value='64'
onclick='cong(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 7 </LABEL><INPUT type='checkbox' name='r' value='128'
onclick='cong(this.value,this.checked);' ><BR>\n");

  statusg=sscanf(data,"l=%2X",&l); //precteni odeslane hodnoty
  printf ("<FORM><LABEL>Hexa Value:</LABEL><INPUT name='l' id='l' value='0'
size='5'><input type='submit' value='SET' ></FORM>\n");
  //zjisteni polohy prepinacu
  printf("</TD><TD width='300px'>");
  //ziskani hexa-kodu GPIO
    unsigned int *swi,stmp,itmp;
    off_t pgoff;
    unsigned char *gpio;
    int fd;
    fd = open("/dev/mem", O_RDWR | O_SYNC);
    pgoff = (unsigned int) (GPIO_BASE_SW & (sysconf(_SC_PAGE_SIZE)-1)); //offset
    gpio = mmap(NULL, sysconf(_SC_PAGE_SIZE), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
GPIO_BASE_SW - pgoff);
    swi = (unsigned int *)(gpio + pgoff);

 //zpracovani hexa-kodu a zobrazeni stavu jednotlivych prepinacu
 ii=131072; //2^(pocet prepinacu-1)
 i=17; //pocet prepinacu-1
 stmp=*swi;
 printf("<H3>SWITCHES STATUS</H3>\n");
 while (i!=-1)
 {
         itmp=ii/2;
         printf("Switch num: %d\n ",i);
         i--;
         if (*swi!=0) {
                 if (stmp!=1) {
                         if (ii<=stmp) {printf("<img src='/green.gif'><br>\n");
                          stmp=stmp-ii;}
                         else { printf("<img src='/red.gif'><br>\n");  }

                  }
                else printf("<img src='/green.gif'><br>");

         }
        else printf("<img src='/red.gif'><br>");
        ii=ii/2;
 }

 printf ("<FORM><input type='hidden' value='1' name='sw'><input type='submit'
   value='RELOAD'></FORM>\n");
   statussw=sscanf(data,"sw=%d",&s);

  ///cervene LED HTML kod///
  printf("</TD><TD>");
  printf("<H3>TURN ON RED LEDs</H3>\n");
    statusr=sscanf(data,"r=%6X",&r);

 /*javascript funkce pro ziskani vysledneho hexa-kodu z kombinace zaskrtnutych
checkboxu */
  printf  ("<script language='javascript'>var gall=0; function con(i,act)
{i=parseInt(i); if (act==1) gall=gall+i; else gall=gall-i;
document.getElementById('allg').value=gall.toString(16);}</script>");
  printf ("<LABEL>LED N. 0 </LABEL><INPUT type='checkbox' name='r' value='1'
onclick='con(this.value,this.checked);' ><BR>\n");
```

```
  printf ("<LABEL>LED N. 1 </LABEL><INPUT type='checkbox' name='r' value='2'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 2 </LABEL><INPUT type='checkbox' name='r' value='4'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 3 </LABEL><INPUT type='checkbox' name='r' value='8'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 4 </LABEL><INPUT type='checkbox' name='r' value='16'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 5 </LABEL><INPUT type='checkbox' name='r' value='32'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 6 </LABEL><INPUT type='checkbox' name='r' value='64'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 7 </LABEL><INPUT type='checkbox' name='r' value='128'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 8 </LABEL><INPUT type='checkbox' name='r' value='256'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 9 </LABEL><INPUT type='checkbox' name='r' value='512'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 10 </LABEL><INPUT type='checkbox' name='r' value='1024'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 11 </LABEL><INPUT type='checkbox' name='r' value='2048'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 12 </LABEL><INPUT type='checkbox' name='r' value='4096'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 13 </LABEL><INPUT type='checkbox' name='r' value='8192'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 14 </LABEL><INPUT type='checkbox' name='r' value='16384'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 15 </LABEL><INPUT type='checkbox' name='r' value='32768'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 16 </LABEL><INPUT type='checkbox' name='r' value='65536'
onclick='con(this.value,this.checked);' ><BR>\n");
  printf ("<LABEL>LED N. 17 </LABEL><INPUT type='checkbox' name='r' value='131072'
onclick='con(this.value,this.checked);' ><BR>\n");


  printf ("<FORM>");
  printf ("<LABEL>Hexa Value:</LABEL><INPUT name='r' id='allg' value='0'
size='5'>\n");
  printf ("<input type='submit' value='SET'></FORM>\n");
  printf("</TD></TR></TABLE>");



  //LCD nastaveni hodnoty (vyuziva znakoveho ovladace LCD16207)
  if (statuslcd){
    FILE *file;
    file = fopen("/dev/lcd16207","w");
    fprintf(file,"%s", m);
    fclose(file);
  }
  //nastaveni zelenych LED (pres mmap, mmu system)
  if (statusg){
    unsigned int *green_led;
    off_t pgoff;
    unsigned char *gpio;
    int fd;
    fd = open("/dev/mem", O_RDWR | O_SYNC);
    pgoff = (unsigned int) (GPIO_BASE & (sysconf(_SC_PAGE_SIZE)-1)); //offset
    gpio = mmap(NULL, sysconf(_SC_PAGE_SIZE), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
GPIO_BASE - pgoff);
    green_led = (unsigned int *)(gpio + pgoff);
    *green_led = 1;
  }
    //nastaveni cervenych LED
  if (statusr){

    unsigned int *red_led;
```

```
    off_t pgoff;
    unsigned char *gpio;
    int fd;
    fd = open("/dev/mem", O_RDWR | O_SYNC);
    pgoff = (unsigned int) (GPIO_BASE_R & (sysconf(_SC_PAGE_SIZE)-1)); //offset
    gpio = mmap(NULL, sysconf(_SC_PAGE_SIZE), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
GPIO_BASE_R - pgoff);
    red_led = (unsigned int *)(gpio + pgoff);
    *red_led = r;
  }

  return 0;
}
```

Příloha H – Ukázkový program na ovládaní led diod pro systém s podporou MMU

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>
#define GPIO_BASE 0x00002420 //Adresa v SOPC bulider

unsigned int *green_led;
off_t pgoff;

int main (void)
 {
 unsigned char *gpio;
 int fd;
 fd = open("/dev/mem", O_RDWR | O_SYNC); //otevri soubor pro zapis i cteni v
asynchronim rezimu
 if (fd < 0) {
   perror("Failed to open /dev/mem");
   return fd;
 }
 pgoff = (unsigned int) (GPIO_BASE & (sysconf(_SC_PAGE_SIZE)-1)); //offset
 gpio = mmap(NULL, sysconf(_SC_PAGE_SIZE), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
GPIO_BASE - pgoff); //namapovani
 if (gpio == MAP_FAILED ) {
   printf("Cannot map registers\n");
 }
 green_led = (unsigned int *)(gpio + pgoff);
 int i,blinks_n,time;
 unsigned short leds;
 printf("LEDs to turn up (hexa value, i.e. FF) \n");
 scanf("%2X",&leds);
 printf("time to keep led on/off (ms)\n");
 scanf("%i",&time);
 printf("number of blinks\n");
 scanf("%i",&blinks_n);

 for (i=0; i<blinks_n; i++)
  {
   *green_led = leds;//nastav zadanou hodnotu
   usleep(1000*time);
   *green_led = 0; //zhasni vsechny led
   usleep(1000*time);
  }
 return 0;
}
```

Příloha I – Znakový ovladač sedmi segmentového displeje

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/ioport.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <asm/io.h>
#include <asm/uaccess.h>

#define na_seven_seg_1 0x00002450 |0xe0000000
// zapsani negovane hodnoty na 7seg
#define WRITE_7(val) outl(~((unsigned short) val), (na_seven_seg_1))

// Cislo prirazene k ovladaci
#define SEDM_SEG_MAJOR 70

// velikost bufferu
#define BUFFER_SIZE 2

// inicializace
char sedm_seg_disp_buf[BUFFER_SIZE] = "\0";
int sedm_seg_disp_device_open = 0;
void write_sedm_seg(unsigned char val);
// Pole s hexa hodnotama pro segemnty
static unsigned char hex_sedm_seg_data[] = {
        0x3F, 0x06, 0x5B, 0x4F, 0x66,
        0x6D, 0x7D, 0x07, 0x7F, 0x6F,
        0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71
};



static int sedm_seg_del(struct inode *inode, struct file *file)
{
        --sedm_seg_disp_device_open;
        module_put(THIS_MODULE);
        return 0;
}

///vytvoreni souboru ovladace
static int sedm_seg_open(struct inode *inode, struct file *file)
{
        //pokud je jiz ovaldac zaregistrovan
        if (sedm_seg_disp_device_open++) {
                printk(KERN_ALERT "Device busy...\n");
                return -EBUSY;
        }
        try_module_get(THIS_MODULE);
        return 0;
}

///Zapis na zarizeni
static ssize_t sedm_seg_write(struct file *filp, const char *buff, size_t len,
loff_t *off)
{
        //kontrola velikosti bufferu
        if (len > (BUFFER_SIZE-1))
                len = BUFFER_SIZE-1;

        // kopirovani hotnot do bufferu
        copy_from_user(sedm_seg_disp_buf, buff, len);
        sedm_seg_disp_buf[len] = '\0';

        // Vystup na segmenty
        write_sedm_seg((unsigned char) sedm_seg_disp_buf[0]);
```

```
        return len;
}


static unsigned long device_get_unmapped_area(struct file *file,
                                              unsigned long addr,
                                              unsigned long len,
                                              unsigned long pgoff,
                                              unsigned long flags)
{
  return pgoff << PAGE_SHIFT;
}

//Operace pro soubor ovladace

///smazani souboru ovaldace
static struct file_operations fops_sedm_seg =
{
        .write   = sedm_seg_write,
        .open    = sedm_seg_open,
        .release = sedm_seg_del,
};

// Zapsani hexa hodnoty na 7seg
void write_sedm_seg(unsigned char val)
{
        unsigned char seg0;
        seg0 = hex_sedm_seg_data[val];


        WRITE_7(seg0);
//        printk(KERN_ALERT "Write to sevenseg %1X ...\n",seg0);
}



//Inicializacni modul

static int sedm_seg_init(void)
{

        // Rezervovani pameti
        if(!request_mem_region((unsigned long) na_seven_seg_1, 8, "sedm_seg"))
                return -1;

        // Registerovani ovladace do kernelu
        if(register_chrdev(SEDM_SEG_MAJOR, "sedm_seg_pio", &fops_sedm_seg))
        {
                printk(KERN_ALERT "Driver register failed ...\n");
                release_mem_region((unsigned long) na_seven_seg_1, 8);
                return -EIO;
        }

        // rozsviceni znaku F pri inicializaci
        write_sedm_seg(0x0F);
        return 0;
}


///Odregistrovani ovladace
static void sedm_seg_exit(void)
{
        unregister_chrdev(SEDM_SEG_MAJOR, "sedm_seg");

        // Uvolneni pameti
        release_mem_region((unsigned long) na_seven_seg_1, 8);
}
```

```
// Vlozeni inicializacni a odregistracni funkce do kernelu
module_init(sedm_seg_init);
module_exit(sedm_seg_exit);
```

Příloha J – Demonstrační aplikace pro okenní manažer Nano-X

```
#define MWINCLUDECOLORS
#include <stdio.h>
#include "nano-X.h"

int main(int ac,char **av)
{
    GR_WINDOW_ID    w;
    GR_GC_ID   gc;
    GR_EVENT   event;
    int       i=5;
    //Otevreni spojeni s Nano-X serverem
    if (GrOpen() < 0) {
        printf("Can't open graphics\n");
        exit(1);
    }
    //Vytvoreni noveho okna na pozici x=100,y=100, s velikosti okna 200x60
    w = GrNewWindow(GR_ROOT_WINDOW_ID, 100, 100, 200, 60,
        4, WHITE, BLUE);
    //Vytvoreni noveho grafickeho obsahu okna
    gc = GrNewGC();
    //nastaveni barvy pozadi
    GrSetGCForeground(gc, BLACK);
    GrSetGCUseBackground(gc, GR_FALSE);
    //Vybereme pouze udalosti, ktere budeme pouzivat, snzime tim zatez na server
    GrSelectEvents(w, GR_EVENT_MASK_EXPOSURE | GR_EVENT_MASK_CLOSE_REQ);
    GrMapWindow(w);

        while (1) {
                //Kazdych 200ms dojde k volani udalosti
                GrGetNextEventTimeout(&event, 200L);
                switch (event.type) {
                      //po vytvoreni okna umistime text na dane souradnice
                       case GR_EVENT_TYPE_EXPOSURE:
                           GrClearWindow(w,0);
                           GrText(w, gc, i, 30, "DEMO TEXT", -1, GR_TFASCII);
                                break;
                        //pri ukonceni aplikace nejprve dojde k ukonceni spojeni se
serverem
                         case GR_EVENT_TYPE_CLOSE_REQ:
                                GrClose();
                                exit(0);
                        //tato udalost je volana kazdych 200ms
                         case GR_EVENT_TYPE_TIMEOUT:
                           i++;
                            if (i==170) i=5;
                           //smazeme obsah okna
                           GrClearWindow(w,0);
                           vykreslime text na novych pozicich
                            GrText(w, gc, i, 30, "DEMO TEXT", -1, GR_TFASCII);
                            break;
                }
        }

    return 0;
}
```

Příloha K – Ukázka práce s přerušením

```
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/init.h>
#include <linux/interrupt.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/ioport.h>
#include <asm/uaccess.h>
#include <asm/io.h>


typedef volatile struct
        {
        int np_piodata;          // read/write, up to 32 bits
        int np_piodirection;     // write/readable, up to 32 bits, 1->output bit
        int np_piointerruptmask; // write/readable, up to 32 bits, 1->enable
interrupt
        int np_pioedgecapture;   // read, up to 32 bits, cleared by any write
        } np_pio;

// PIO Routines
void nr_pio_showhex(int value); // shows low byte on pio named na_seven_seg_pio


#define na_button 0xe0002460 //base address + MMU
#define na_button_irq 6
#define green_led (unsigned short*) 0xe0002420 //base address LEDs + MMU
#define BUTTONS_MAJOR       244 //hlavni cislo

#define BUFFER_SIZE      3

char buttons_string[BUFFER_SIZE] = "\0";
int buttons_is_open = 0;

int *devid = (int*) na_button;


//handler preruseni
static irqreturn_t button_isr(int irq, void *dev_id, struct pt_regs *regs)
{
    int status = 0;
    unsigned char orig;
    np_pio* buttons_pio = (np_pio *)na_button;

    //zachyceni preruseni
    outl(0, &buttons_pio->np_pioedgecapture);

    // zakazani preruseni
    outl(0, &buttons_pio->np_piointerruptmask);

    //ktere tlacitko bylo stisknuto
    status = (inl(&buttons_pio->np_piodata)) & 0xF;

    printk("button pressed: %X\n",status);

    *green_led=0;
    *green_led=status;
    //povoleni preruseni
    outl(-1, &buttons_pio->np_piointerruptmask);
    return IRQ_HANDLED;
}

//funkce spustena pri inicalizaci
static int button_start(int *dev_id)
{
```

```
    np_pio *buttons_pio = (np_pio *)(na_button);

    // nastaveni pinu na vystupni
    outl(0, &buttons_pio->np_piodirection);
    // vycisteni aktivnich preruseni
    outl(0, &buttons_pio->np_pioedgecapture);

    // rutina pro obsluhu preruseni (ISR)
    if (request_irq(na_button_irq, button_isr, 0, "buttons", (void *) (dev_id))) {
        printk("de2_buttons: unable to register interrupt %d\n", na_button_irq);
        return -1;
    }

    // aktivovani preruseni na vsech pinech
    outl(-1, &buttons_pio->np_piointerruptmask);

    buttons_string[BUFFER_SIZE] = "\0";
    return 0;
}


static int button_open(struct inode *inode, struct file *filp)
{
    if (buttons_is_open++) {
        printk("device busy...\n");
        return -EBUSY;
    }

    try_module_get(THIS_MODULE);
    return 0;
}

///Handler pro stisknuti/uvolneni tlacitka
static int button_release(struct inode *inode, struct file *filp)
{
    --buttons_is_open;
    module_put(THIS_MODULE);
    return 0;
}

// Structure which assigns handler functions to hardware operations
static struct file_operations fops_buttons = {
    .open   =   button_open,
    .release=   button_release,
};

//inicizlizacni funkce
static int __init button_init(void)
{
    // rezervovani pameti
    if(!request_mem_region((unsigned long)na_button, sizeof(np_pio), "buttons"))
        return -1;

    // registrovani znakoveho olvadace do jadra
    if(register_chrdev(BUTTONS_MAJOR, "buttons", &fops_buttons))
    {
        printk("register_chrdev of de2_buttons failed!\n");
        release_mem_region((unsigned long)na_button, sizeof(np_pio));
        return -EIO;
    }

    button_start(devid);
    return 0;
}

//ukoncujici funkce
static void __exit button_exit(void)
{
```

```
    np_pio *buttons_pio = (np_pio *)(na_button);
    // zakazani preruseni
    outl(0, &buttons_pio->np_piointerruptmask);

    // odregistrovani preruseni
    free_irq(na_button_irq, (void *) (devid));

    // odregistrovani znakoveho zarizeni
    unregister_chrdev(BUTTONS_MAJOR,"buttons");

    // uvloneni pameti
    release_mem_region((unsigned long)na_button, sizeof(np_pio));
}
module_init(button_init);
module_exit(button_exit);
```