Evropská unie
Evropský fond pro regionální rozvoj
Partner pro váš rozvoj

ROP Střední Morava

Zlínský kraj

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

Embedded systém s OS Linux

Závěrečná zpráva k inovačnímu voucheru

Ing. Josef Hájek

OBSAH

ÚVOD

Cílem tohoto projektu bylo vytvoření prototypu embedded systému s OS Linux pro řízení a monitoring distribučních sítí. Typické nasazení systému jsou např. distribuční body rozlehlé počítačové sítě (WAN), kde je potřeba monitorovat funkci jednotlivých datových spojů, v případě výpadku daný spoj restartovat, poslat email/SMS zprávu pracovníkům dohledu atd.

Většina podobných systémů, které jsou v současné době na trhu dostupné, je založena na slabších procesorech ARM Cortex M, které neumožňují nasazení OS Linux a v aplikacích s připojením k sítím TCP/IP mají mnoho omezení v oblasti výkonu i bezpečnosti.

Důvodem pro výběr platformy Linux/ARM je obrovská aplikační flexibilita se zachováním nízké spotřeby a možnost nasazení ve venkovních instalacích s širokým rozsahem pracovních teplot.

Realizace systému proběhla ve 3 fázích:

1. Analýza dostupných CPU platforem,
2. Návrh a implementace desky plošných spojů,
3. Implementace OS Linux na této platformě.

Hlavním řešitelem tohoto projektu byl Ing. Josef Hájek. Na implementaci web rozhraní, založeného na systému Freenetis, spolupracovali studenti Bc. Ondřej Fibich, David Raška a Jan Dubina.

Popis realizace celého projektu je uveden v následujících kapitolách.

1 TESTOVANÉ PLATFORMY

V rámci úvodní analýzy jsme otestovali 2 procesorové moduly, které se lišily prakticky ve všech ohledech, s výjimkou společné architektury. Pro každý modul je dále uvedena tabulka, která shrnuje základní informace o modulu. Kromě těchto 2 modulů jsme testovali také Linux na modulu Colibri T20 (Nvidia Tegra 2), desce Leopardboard (TMS320DM365) a desce Beagleboard (AM37x).

Cílem práce bylo zprovoznit na nějakém modulu všechny požadované periferie, mezi které patřily: síťová karta, SD karta, USB sběrnice, LVDS LCD displej, hardwarová GPIO klávesnice a GPIO pro ovládání silových vstupů a výstupů.

Na modulu Colibri PXA320 bylo dosaženo všech požadavků s výjimkou funkční síťové karty v Linuxu a hardwarové akcelerace LCD displeje, proto jsme použili modul TQMa53, u kterého výrobce deklaroval většinu požadovaných vlastností jako funkčních. Tento modul navíc má vestavěný USB a CAN oproti modulu Colibri T20, u kterého je použit stejný Ethernet čip, jako měl modul Colibri PXA320, a který při vyšších přenosových rychlostech trpěl chybami.

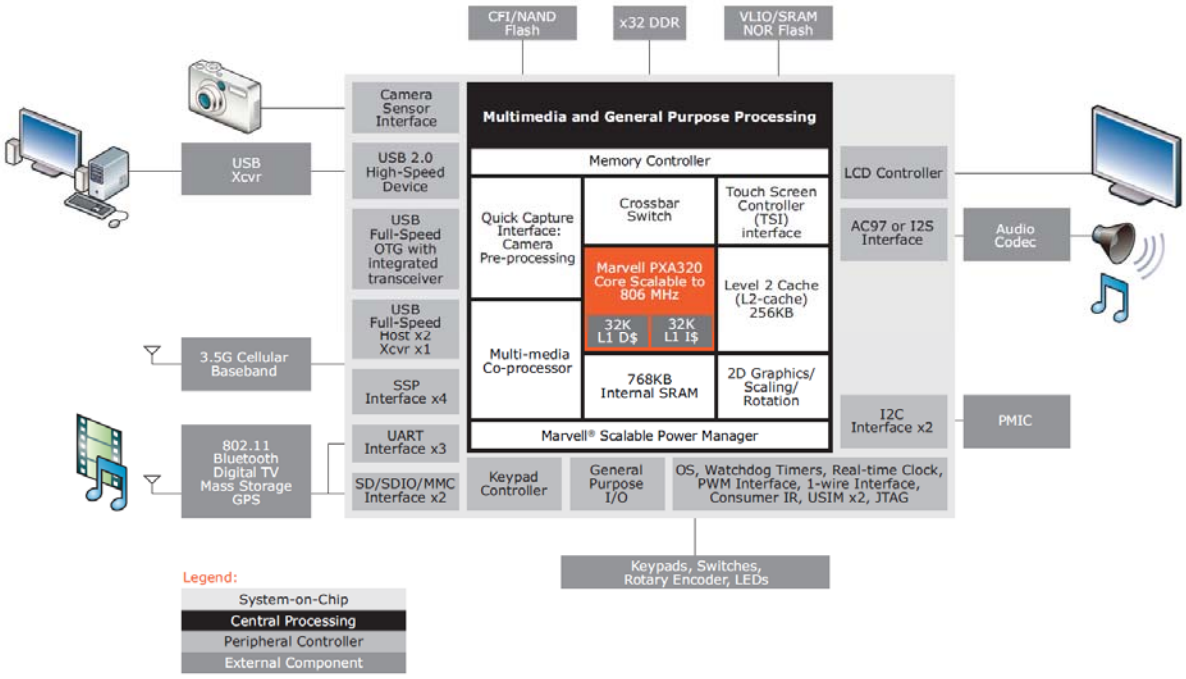
1.1 Modul Colibri PXA320

Název modulu:	Colibri PXA320
Výrobce modulu:	Toradex AG
Procesor:	Marvell PXA320
Nosná deska:	Toradex Colibri Evaluation Board
Použitý zavaděč:	U–boot
Použitý nástroj pro vytvoření systému:	Buildroot

Tabulka 1 – Základní informace o modulu Colibri PXA320

Procesor PXA320 od firmy Marvell je pokračovatelem starších procesorů řady StrongARM od Intelu. Jeho jádrem je XScale – implementace ARMv5TE architektury. Jeho sériová výroba začala již v roce 2006, ale do dnešních dnů je stále k dostání a je garantována dostupnost modulů s tímto procesorem až do roku 2017.

Maximální taktovací frekvence tohoto procesoru je 806MHz, má 256KB L2 cache a 32-bitové DDR rozhraní. Na následujícím obrázku Obrázek 1 – Diagram procesoru Marvell PXA320 jsou znázorněny rozhraní a části tohoto procesoru.



Obrázek 1 – Diagram procesoru Marvell PXA320

Při implementaci Linuxu jsme použili zavaděč U-boot, protože byl jediný, který podporu tohoto modulu a nosné desky obsahoval. Jako nástroj pro tvorbu systému byl vybrán Buildroot.

Výrobce modulu dává na svém webu k dispozici upravené linuxové jádro, které je však bohužel poměrně zastaralé a podobně je na tom i poskytovaný zavaděč. Do novějších řad jádra se sice dostala podpora tohoto modulu, ale některé části (například podpora síťového čipu pro novější revize modulu) stále chybí a vzhledem ke stáří modulu a čipu je vysoce nepravděpodobné, že by byla ještě někdy přidána.

Tento modul jsme otestovali s vývojovou deskou od výrobce modulu.

Výsledek testů je ten, že tento modul nelze doporučit pro vývoj nových zařízení s Linuxem. Pro novější revize modulu není dostupná kompletní podpora v novějších linuxových jádrech ani v zavaděči.

1.2 Modul TQMa53

Název modulu:	TQMa53
Výrobce modulu:	TQ Group GmbH
Procesor:	Freescale i.MX53
Nosná deska:	MBa53
Použitý zavaděč:	Barebox
Použitý nástroj pro vytvoření systému:	PTXdist

Tabulka 2 – základní informace o modulu TQMa53

Pro modul TQma53 dodává výrobce již předpřipravený balíček s podporu desky (BSP) pro PTXdist, proto nebylo nutné dělat mnoho úprav – pouze upravit výběr instalovaného softwaru a upravit DeviceTree soubory tak, aby odpovídali všemu připojenému hardwaru.

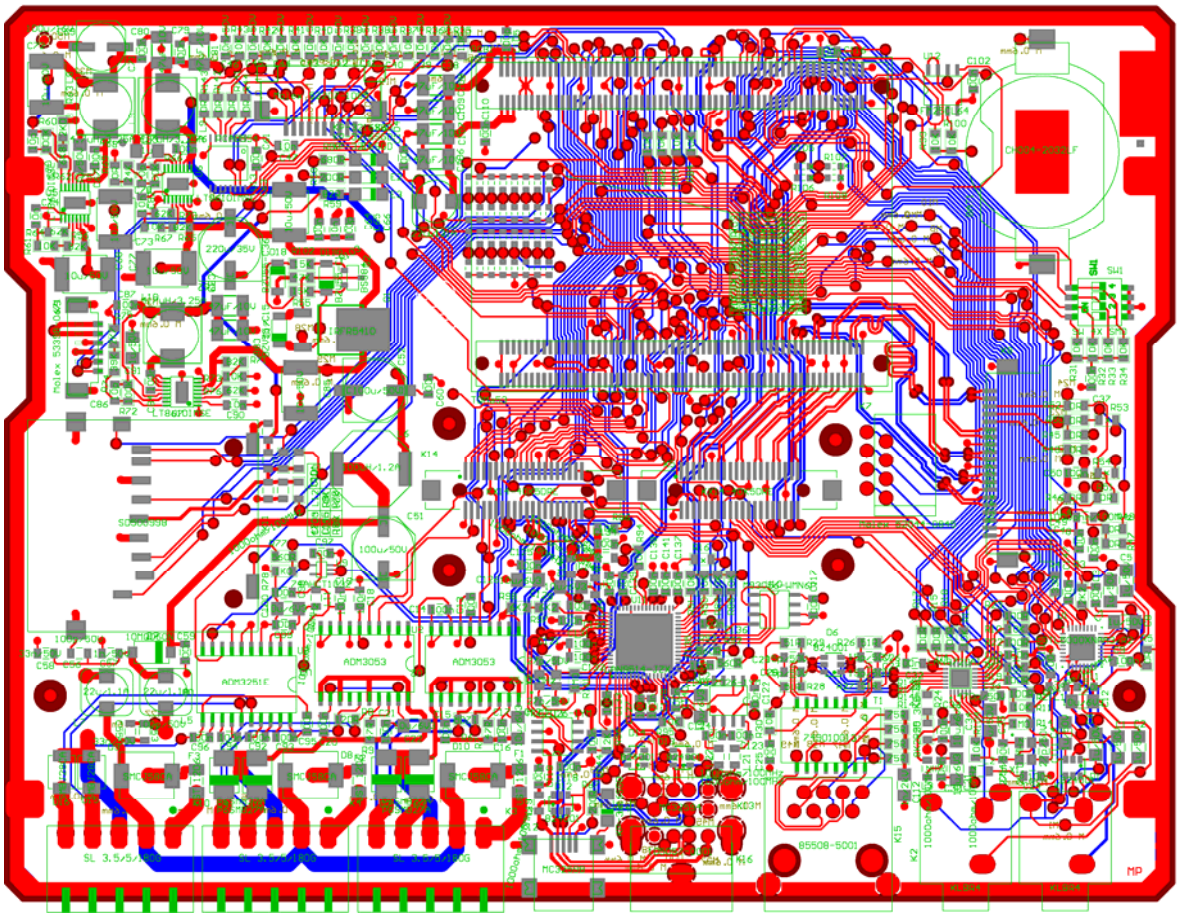
Jako nosná deska pro tento modul byla použita deska od výrobce modulu MBa53.

Výrobce tohoto modulu – firma TQ Group GmbH – nakonec v nedávné době zařadil do nejnovějšího balíčku pro podporu desky nejnovější jádro a spolu s ním i ovladač CODA, který obsahuje hardwarovou akceleraci zobrazování na LCD displeji. Proto jsme se rozhodli implementovat celý projekt s procesorem i.MX53.

Obrázek 2 – pohled na modul TQMa53 na desce MBa53

2 NÁVRH DESKY PLOŠNÝCH SPOJŮ

Návrh desky plošných spojů pro modul TQMa53 byl relativně jednoduchý, protože zapojení většiny potřebných periferií je zdokumentováno ve schematu vývojové desky MBa53. Výsledný design je na obrázku níže:



Obrázek 3 – Plošný spoj základní desky pro CPU modul TQMa53

Na desce jsou kromě napájecího zdroje (DC/DC převodník) obsaženy všechny požadované periferie – Ethernet, RS232, USB, logické vstupy chráněné optočleny, 4 silové výstupy (2x relé a 2x MOSFET) pro spínání zátěže např. připojení/odpojení napětí zatuhlých síťových zařízení.

Po osazení napájecího zdroje, RS232 převodníku a konektoru pro MicroSD kartu jsme ověřili, že modul bootuje. Následně byl osazen a oživen Ethernet. Pak mohla začít fáze 2 – implementace OS Linux na desku.

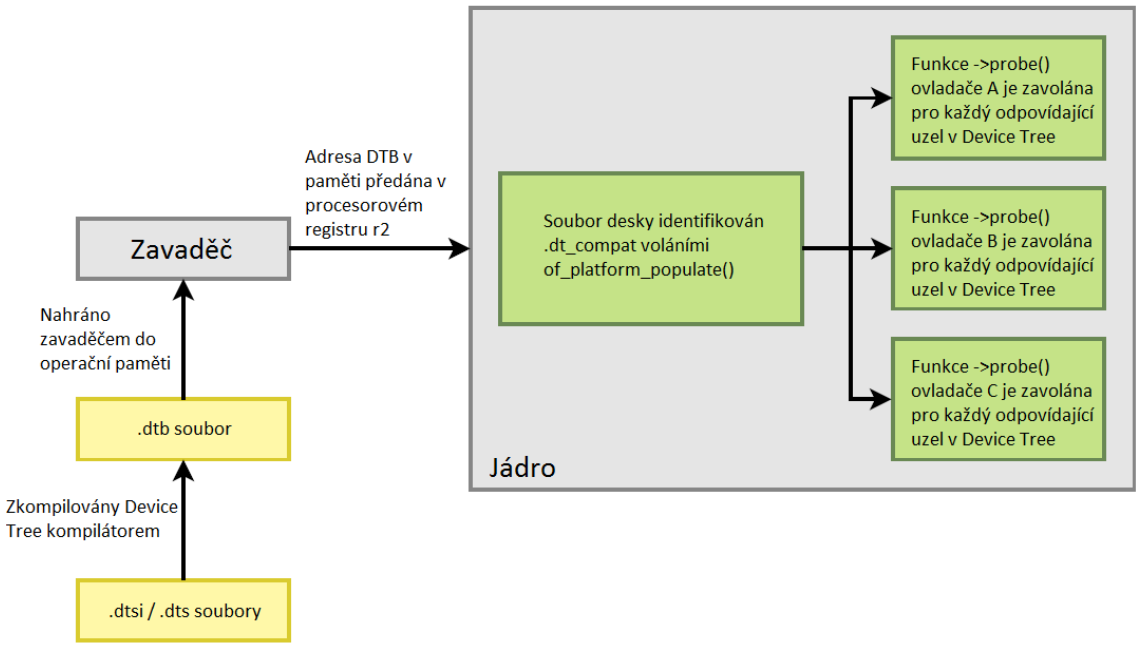
3 IMPLEMENTACE LINUXU NA NOVÉ DESCE

Pro implementaci Linuxu na nový HW je nutné znát způsob, jakým se dnešní jádro ARM Linuxu váže na konkrétní HW zařízení. Toto je popsáno v následujících podkapitolách.

3.1 Device Tree

V minulosti bylo nutné pro každou novou desku napsat specifický kód, který byl pro všechny desky velmi podobný a málo přehledný. Device Tree je způsob, jak přesunout specifické detaily pro jednotlivé desky do člověkem čitelných souborů a ponechat ve zdrojových kódech pouze obecnou inicializaci, která při bootu využije předaný device tree soubor obsahující detaily o platformě.

Device Tree obsahuje stromovou strukturu komponent, díky které je možné určit, jak je která komponenta zapojená na desce, jaké využívá adresní rozsahy, s jakým hardwarem je kompatibilní a podobně. Zdrojový device tree soubor (Device Tree Source) se před použitím musí zkompilovat pomocí device tree kompilátoru (Device Tree Compiler) a výsledkem je binární soubor device tree (Device Tree Blob). Tento soubor pak při bootu musí zavaděč předat jádru. Existují také Device Tree Source Include, které slouží k uchovávání společných částí mezi různými deskami založenými například na stejném procesoru. [10]



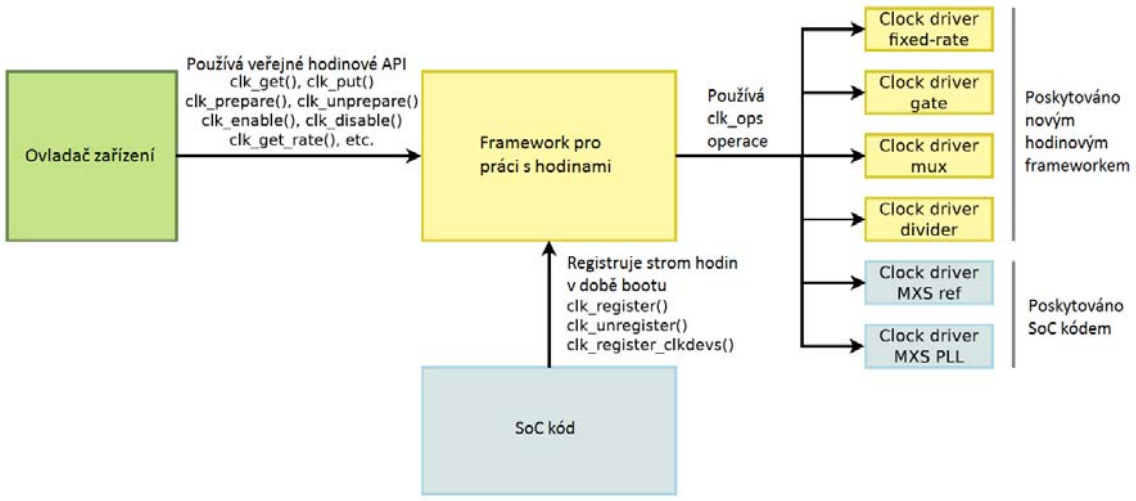
Obrázek 4 – inicializace s pomocí Device Tree [10]

3.2 Nový framework pro práci s hodinami

Různé hardwarové části integrovaného systému jsou řízené různými hodinami, které pracují na různých frekvencích. Většina těchto hodin je součástí komplikovaného stromu, ve kterém jsou rodičovské hodiny vstupem do potomků. Velká část těchto hodin je softwarové konfigurovatelná (zapnutí/vypnutí, změna frekvence) a musí být možné s nimi za běhu manipulovat pro podporu různých šetřících režimů.

Ve starších jádrech byl seznam hodin definován a kontrolován kódem pro podporu dané platformy. Existovala sice struktura pro ovládání hodin a společné API, ale obojí bylo pro každou subarchitekturu definováno a implementováno jinak, i když mezi různými subarchitekturami byly pouze malé rozdíly.

Do linuxového jádra byl s verzí 3.4 poprvé přidán nový framework pro manipulaci s hodinami, který implementuje společné API a definuje datové struktury, pomocí kterých si může podpůrný kód definovat svoje hodiny. [10]



Obrázek 5 – použití nového frameworku pro práci s hodinami [10]

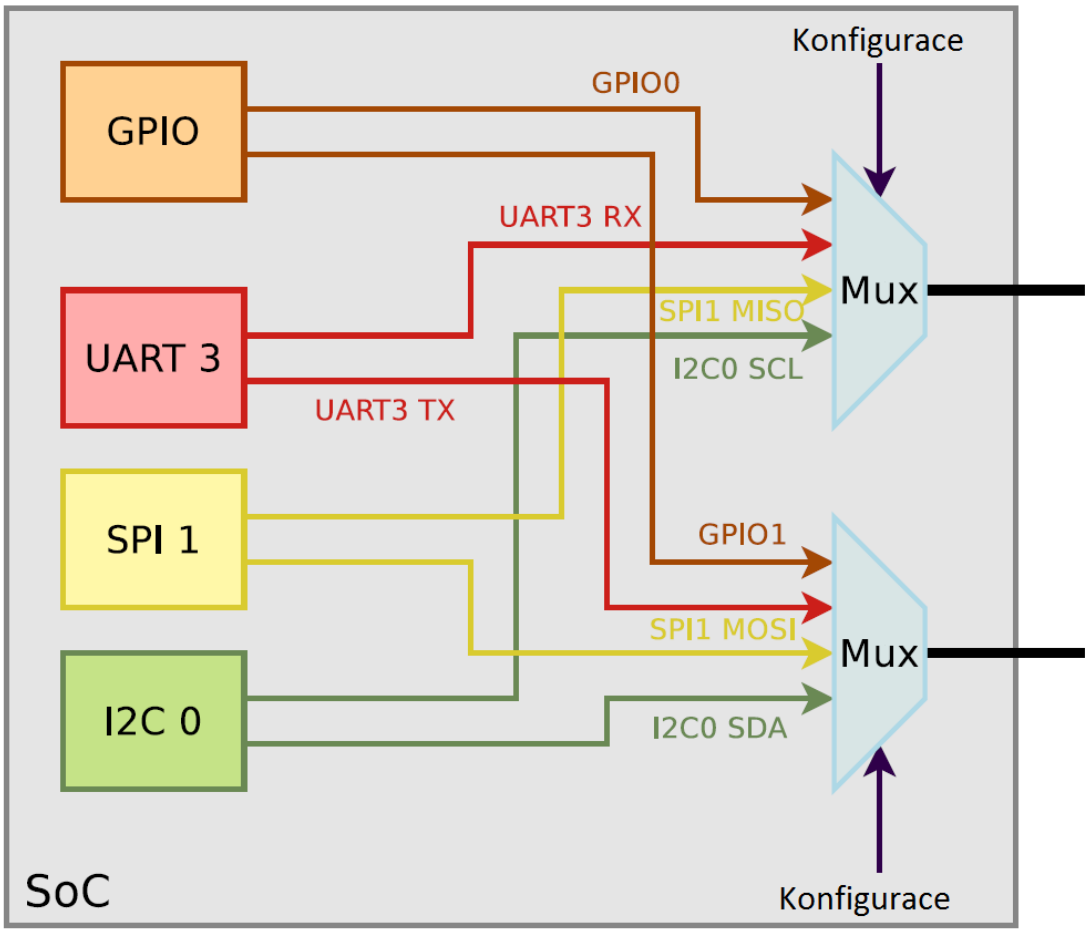
3.3 Pin muxing

Dnešní integrované čipy obsahují velké množství periferií, jejichž množství může přesahovat počet dostupných pinů. Z tohoto důvodu jsou tyto piny multiplexovány – mohou být použity jako funkce A, B, C, nebo GPIO.

Příkladem těchto funkcí jsou:

- SDA/SCL vodiče pro I2C sběrnice
- MISO/MOSI/CLK vodiče pro SPI
- RX/TX/CTS/DTS vodiče pro UARTy

Na obrázku Obrázek 6 je znázorněn princip multiplexování pinů procesoru.



Obrázek 6 – princip multiplexování pinů procesoru [10]

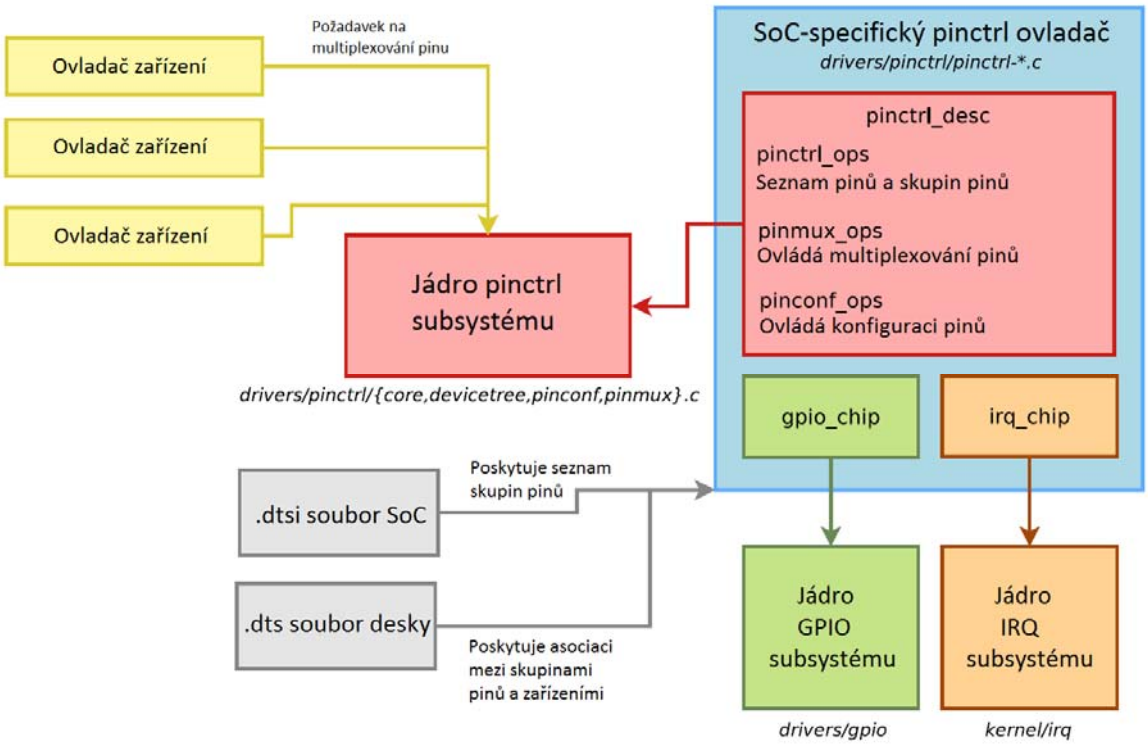
V jakém módu bude každý pin použit je konfigurováno softwarově a závisí to na tom, jak bude tento čip zapojen na nosné desce. V linuxovém jádře měla každá ARM subarchitektura vlastní kód starající se o multiplexování pinů. API tohoto kódu proto bylo pro každou subarchitekturu specifické a multiplexování pinů muselo být nastavováno v kódu pro podporu procesoru a nemohlo být měněno z ovladačů.

Problémy s tímto přístupem se snaží řešit nový pinctrl subsystém, jehož hlavním autorem je Linus Walleij, z firmy Linaro/ST-Ericsson. Implementaci tohoto subsystému je možné nalézt v *drivers/pinctrl* a poskytuje následující:

- API pro registraci pinctrl ovladače pro entity které mají k dispozici seznam pinů, jejich funkcí a vědí jak je konfigurovat. Používáno ovladači specifickými pro SoC ke zveřejnění možností multiplexování pinů.

- API pro ovladače zařízení dovolující požádat o multiplexování určité sady pinů
- Interakci s GPIO frameworkem

Následující obrázek znázorňuje funkci pinctrl subsystému.



Obrázek 7 – funkce pinctrl subsystému [10]

3.4 Křížový překladač

Křížový překladač (v angličtině Cross-Compiler) je takový překladač, který generuje spustitelný kód pro jinou platformu, než na které je překladač spuštěn. Mezi křížové překladače se řadí všechny překladače, které jsou distribuovány ve vývojových prostředích od jednotlivých výrobců mikropočítačů – například Code Composer Studio, IAR Workbench, Tasking a jiné.

Pro zkompilování zavaděče, linuxového jádra a zbytku systému však nejsou tyto překladače vhodné, neboť zkompilování těchto částí zpravidla s těmito překladači nikdo netestoval a navíc část zdrojových kódů obsahuje použité neoficiální rozšíření jazyka, která jsou součástí kompilátoru obsaženým v GNU Compiler Collection (GCC). Z tohoto důvodu je v současné době nejvhodnější použít křížový překladač založený na GCC.

Buď je možné použít některý z předpřipravených křížových překladačů, například CodeSourcery ARM GNU/Linux tool chain nebo si vytvořit vlastní křížový překladač, například pomocí nástroje Crosstool-ng. Obecné předpřipravené překladače mají tu výhodu, že jsou otestované a rychle a snadno dostupné. Na druhou stranu ale kvůli podpoře co nejvíce modelů procesorů nevyužívají všechny možnosti procesoru, mohou obsahovat podporu pro nepotřebné věci nebo jim naopak může chybět podpora pro něco potřebného.

Proto jsou předpřipravené překladače vhodné spíš jako rychlá testovací možnost, ale cílový systém by měl být zkompilován překladačem na míru.

3.5 Zavaděč

Zavaděč (v angličtině bootloader) je malý program, jehož úkolem je inicializovat minimum zařízení připojených k procesoru a následně zavést další systém (Linux, WinCE, atd.).

U stolních/přenosných počítačů založených na platformě x86 je na základní desce v EEPROM či jiné paměti uložený BIOS případně EFI/UEFI. Ten detekuje bootovací zařízení a z něj spouští zavaděč dalšího systému, případně může rovnou zavést daný systém. Na vestavěných zařízeních naproti tomu úlohu BIOSu přebírá zavaděč. Konkrétní bootovací sekvence navíc závisí na typu použité paměti.

V případě, že je zavaděč uložen v paměti NOR, může procesor spustit zavaděč přímo odtamtud. V případě NAND pamětí je situace komplikovanější, protože kód uložený v NAND paměti nelze přímo spustit a musí být nejdříve nahrán do RAM paměti. Procesor ale nemá dopředu informaci o tom, jak velký zavaděč je, proto NAND paměť obsahuje v několika prvních blocích ještě další předzavaděč, který zavádí buď plný zavaděč, nebo rovnou následující systém. Tento předzavaděč se označuje různými termíny, například Initial Program Loader (IPL) nebo Secondary Program Loader (SPL). Předzavaděč už má v sobě uloženou informaci, jak velký je plný zavaděč nebo jádro systému a proto ho může zkopírovat do RAM a opět na něj přeskočit.

Celý postup lze tedy v případě NAND pamětí shrnout takto:

- Procesor obsahuje v permanentní paměti krátký kód, který z NAND paměti překopíruje prvních několik bloků NAND paměti, ve kterých je uložený předzavaděč, do RAM paměti a provede JMP na nahraný kód.
- Předzavaděč zkopíruje další zavaděč nebo jádro systému do RAM a provede JMP.
- V případě, že byl v předchozím kroku zkopírován zavaděč, se v něm provede inicializace nutných periferií a poté se zavede systém. Další systém může být přitom zaváděn z SD karty, USB zařízení nebo třeba sítě.

Ne pro všechny platformy existuje takový předzavaděč, který je schopný přímo zavést jádro systému, proto je tento mezikrok často nutný, i když je systém uložený v NAND paměti.

Pro použití na vestavěných systémech existují pouze 3 univerzální zavaděče – Das U-boot, Barebox a RedBoot. Ani pro jeden testovaný modul nebyla v RedBootu podpora, proto jsou dále popisovány pouze zavaděče Das U-boot a Barebox.

3.5.1 Das U-boot

Das U-boot (Universal Bootloader) je opensource zavaděč pro použití ve vestavěných systémech. Je dostupný pro velké množství různých architektur. Při spuštění a přerušení bootování U-boot přejde do CLI režimu, ve kterém je možné provádět různé jednoduché činnost jako je aktualizace zavaděče/jádra/systému, úprava bootovací sekvence a podobně. Toto prostředí obsahuje proměnné prostředí, které se navíc mohou chovat jako jednoduché skripty – například je možné vytvořit skript složený ze 4 příkazů, který automaticky stáhne novou verzi systému z TFTP serveru a přepíše původní systém v NAND paměti. Kromě vlastních proměnných, které je možné jednoduše nadefinovat buď za běhu nebo již při přípravě zavaděče, obsahuje U-Boot také několik proměnných, které mají speciální význam a které je nutné mít nastaveny pro správnou funkci, případně nabootovaní dalšího systému. V příloze Příloha I: Příkazy U-bootu lze nalézt seznam příkazů podporovaných v CLI U-bootu včetně popisu.

3.5.2 Barebox

Barebox je zavaděč původně odvozený od zavaděče U-Boot, který se ale snaží o stejný styl kódu a kvalitu, jako má linuxové jádro. Přestože je Barebox odvozený z U-Bootu, nemá

podporu pro tak velké množství procesorů/desek/zařízení. Příloha P II:Příkazy Bareboxu obsahuje seznam příkazů použitelných v CLI Bareboxu.

Následující seznam obsahuje některé klíčové vlastnosti, které odlišují Barebox od ostatních dostupných zavaděčů.

- **POSIXové souborové API**

Barebox využívá široce uznávané funkce open/close/read/write/lseek spolu s modelem reprezentování zařízeí pomocí souborů. Díky tomu je API důvěrně známe komukoliv se zkušenostmi s programováním pod Unixovými systémy.

- **Konzole**

Konzole v Bareboxu obsahuje standardní příkazy jako je ls/cd/mkdir/echo/cat,…

- **Souborový systém prostředí**

Na rozdíl od U-Bootu Barebox nezneužívá proměnné prostředí k vytváření skriptů. Po spuštění zavaděče se uživateli nasktne pohled na konzoli a něco, co vypadá jako souborový systém. Ve skutečnosti je to ale jednoduchý ar archív, který je příkazem loadenv načtený z flash paměti do ramdisku a příkazem saveenv uložený zpět.

- **Podpora souborových systémů**

Při spuštění je souborový systém prostředí (env) připojen do / a souborový systém zařízení připojený do /dev, aby bylo možné přistupovat k jednotlivým zařízením. Další souborové systémy mohou být připojeny podle potřeby.

- **Model ovladačů (zapůjčený z Linuxu)**

Barebox následuje linuxový ovladačový model: zařízení mohou být specifikována v hardwarově specifickém souboru a ovladače jsou zodpovědné za zařízení pokud mají stejné jméno.

- **Zdroj hodin**

Používá se stejné API pro práci se zdroji hodin jako v Linuxu.

- **Kconfig/Kbuild**

K sestavení Bareboxu se používá systém Kconfig/Kbuild známý z linuxového jádra, který uložňuje paralelizované sestavování a odstraňuje nutnost mít v kódu hodně ifdefů.

- **Sandbox**

Při vyvíjení Bareboxu je možné sestavit Barebox jako 'sandbox', což zkompiluje Barebox jako standardní POSIXovou aplikaci pro Linux. Tato aplikace může být spuštěna jako normální příkaz a dokonce má i přístup k síti. Soubory z lokálního souborového systému mohou být použity k simulaci zařízení.

- **Parametry zařízení**

Barebox obsahuje model parametrů pro zařízení – každé zařízení může specifikovat vlastní parametry a tyto parametry existují pro každou instanci tohoto zařízení. Parametry mohou být měněny na příkazové řádce ve stylu \<devid\>.\<param\>=”…”. Například změnu IPv4 adresy síťové karty zastupované zařízením eth0 je možné provést příkazem *'eth0.ip=192.168.0.7*' a *'echo $eth0.ip'*.

- **Getopt**

Barebox má odlehčenou implementaci funkce getopt(). Díky tomu není nutné identifikovat parametry příkazů jen podle pozice, což může mít negativní vliv na čitelnost.

- **Integrovaný editor**

Skripty mohou být upravovány malým integrovaným fullcreen editorem. Tento editor má pouze nutné minimum funkcí: posouvání kurzoru, vkládání znaků, uložení a ukončení.

3.6 Linuxové jádro

Linuxové jádro je možné sestavit ručně, nebo ho nechat sestavit v rámci nástrojů pro tvorbu embedded systému, které jsou popsány v kapitole Nástroje pro tvorbu systému.

Ve všech případech však musí být daný procesor + periferie podporovány buď přímo v jádře, nebo musí existovat patche, které tuto podporu přidávají.

Nastavení linuxového jádra závisí z velké části na hardwaru, na kterém bude provozováno a na softwaru, který bude na platformě instalován. Pouze výjimečně se ve vestavěných

systémech používá systém modulů, neboť jádro zpravidla nepotřebuje podporovat více různých konfigurací a je na míru vytvořeno danému hardwaru.

3.6.1 Adresářová struktura jádra pro podporu platformy ARM

Hlavní adresář obsahující zdrojové kódy pro platformu ARM je *arch/arm*. V tomto adresáři je několik souborů a podadresářů, které obsahují hlavičkové soubory, zdrojové soubory a další.

Význam a umístění důležitých souborů a adresářů:

Kconfig – konfigurační soubor pro nástroj Kconfig, který popisuje jednotlivé volby pro platformu ARM

Makefile – soubor obsahující instrukce pro pro sestavovací systém make

boot – obsahuje podpůrné nástroje jádra týkající se bootování

boot/dts – obsahuje Device Tree Source (dts) a Device Tree Source Include(dtsi) soubory, které popisují hardwarové zapojení desek, její periferie a podobně

configs – obsahuje předpřipravené konfigurace jádra pro specifické platformy, v budoucnosti pravděpodobně dojde k odstranění/neaktualizování tohoto adresáře

mach-<název_subarchitektury> – obsahuje soubory specifické pro danou subarchitekturu – podpora procesoru, správu napájení, správu paměti, časovače, obecná inicializace desky,… Konkrétní uspořádání tohoto adresáře se značně liší.

plat-<název> – některé podobné subarchitektury využívají plat-<název> pro společný kód

3.6.2 Podpora starších subarchitektur

Pro starší a již málo používané subarchitektury je zachován systém, při kterém je veškerý popis hardwaru na desce realizovaný v C souboru pro danou desku. Problém s tímto přístupem je ten, že při jakékoliv změně zapojení, nebo třeba výměně jednoho z čipů, je nutné provádět úpravy ve zdrojových kódech. Navíc v případě 2 totožných desek s například různým síťovým řadičem je nutné buď brát každou desku jako samostatnou platformu, nebo do souboru přidat preprocesorové makra pro kontrolu kompilace, které mají za následek výrazně horší čitelnost kódu a náchylnost k chybám. Navíc způsobují

rychle rostoucí komplexitu konfigurace jádra. Problém je rovněž s procesorovými moduly, které nejsou v návrhu zohledněny vůbec a jejichž použití má podobné následky.

3.6.3 Podpora novějších subarchitektur

Pro novější subarchitektury se používá struktura Flattened Device Tree (FDT). Kromě zredukování duplikovaného kódu a postupné standardizaci není díky FDT nutné dělat pro podporu nových desek zásahy ve zdrojovém kódu jádra. Podpora FDT není hotová pro všechny ARM aubarchitektury – pouze pro ty aktuálně používané a nově přidávané.

3.6.4 Porovnání obou přístupů

Pro porovnání je použita ARM subarchitektura nVidia Tegra, která ve starších jádrech je tvořena starším způsobem se zvláštními soubory se zdrojovým kódem pro každou podporovanou desku a v novějších jádrech již využívá Device Tree. Implementace této subarchitektury je v jádře v adresáři *arch/arm/mach-tegra*.

Pro porovnání bude jako starší jádro sloužit verze 3.0.71 a jako novější bude nejnovější mainline – 3.9-rc5. Následující tabulka srovná obě jádra z pohledu podpory různých desek.

Kritérium	3.0.71	3.9-rc5
Počet souborů v *arch/arm/mach–tegra*	44	50
Z toho soubory pro podporu konkrétních desek	14	3
Počet podporovaných desek	6	18

Tabulka 3 – porovnání starší a novější implementace subarchitektury

Při porovnání počtu souborů je pak patrné, že mezi těmito verzemi jádra ubylo 11 souborů pro podporu konkrétních desek a přibylo 17 souborů vylepšující podporu této subarchitektury (podpora uspávání, řízení frekvence procesorů a další.)

3.7 Mechanismus snapshotů

Tento mechanismus, někdy nazývaný suspend to disk nebo hibernace, umožňuje snížit dobu načítání systému tím, že se při vypínání systému uloží funkční stav a při startu systému se tento snapshot načte zpět do paměti. Množství ušetřeného času silně závisí na

velikosti nasazeného systému, rychlosti permanentní paměti a rychlosti procesoru. Linux obecně tento přístup v hlavní větvi jádra na platformě ARM nepodporuje a důvodů pro to je několik. Prvním důvodem je nutnost spolupráce ovladačů hardwaru se systémem na vytváření snapshotů. Ovladače musí podporovat uložení aktuálního stavu a zpětně obnovení bez nutnosti provádět kompletní inicializaci. Pokud se obnovení ze snapshotu provádí až v linuxovém jádře, je ušetřený čas poměrně nízký, protože se před obnovením ze snapshotu musí inicializovat ovladače. Řešením je provádět obnovení ze snapshotu už ze zavaděče, ale to opět vyžaduje určitou minimální inicializaci ovladačů.

I přes problémy, které obnovení ze snapshotu provázejí je zdokumentováno několik případů, kdy se pomocí systému snapshotů na různých zařízeních dosáhnout významné úspory času. Bez vyjímek ale na zprovoznění pracovali lidé s vysokými zkušenostmi s linuxovým jádrem, protože jsou nutné poměrně velké změny v jádře, jaderných ovladačích a zavaděči. [23]

Pro starší linuxová jádra jsou k dispozici sady patchů, které tuto funkcionalitu do jisté míry přidávají, ale jejich funkčnost je závislá na konkrétním hardwarovém vybavení a použitých ovladačích. Tyto patche lze najít pod názvem *swsusp* nebo *suspend2* for ARM. [24]

Navíc je možné použít řešení QuickBoot od firmy Ubiquitous, které je ale komerční. [25]

4 LINUXOVÉ NÁSTROJE

Tato kapitola je zařazena z toho důvodu, že pro použití jakéhokoliv nástroje pro vytvoření linuxového systému s nimi uživatel pravděpodobně přijde do styku. Tato část však ke čtení předpokládá určitou zkušenost s Linuxem a programováním, proto jsou popsány pouze ne úplně běžné nástroje.

4.1 Konfigurační nástroj Kconfig

Kconfig je konfigurační mechanismus, který původně vznikl ke konfiguraci linuxového jádra a který postupně adaptovali další open source projekty jako je BusyBox, Buildroot, crosstools–ng nebo uClibc. Tento mechanismus vytváří stromovou strukturu konfiguračních voleb, mezi kterými mohou být různé vzájemné závislosti a kterými je možné nakonfigurovat například linuxové jádro přesně na míru.

Existuje více variant konfiguračních „frontendů“ jak pro konzoli (menuconfig) tak pro grafické prostředí (xconfig). Výstupem konfigurace je jeden soubor, který má běžně název .config. V něm jsou veškeré volby, které byly pomocí konfiguračního nástroje nastaveny.

Příklad volby nastavené na ano:

CONFIG_ARM=y

Příklad volby nastavené na ne:

 # CONFIG_MODULE_FORCE_LOAD is not set

Příklad volby nastavené na hodnotu:

CONFIG_INIT_ENV_ARG_LIMIT=32

Z tohoto konfiguračního souboru je pak ještě možné vyrobit takzvaný defconfig, což je soubor, který má stejnou syntaxi, pouze jsou vynechány volby, které jsou nastavené stejně, jako je výchozí stav (jsou redundantní).

Jedna z výhod Kconfigu je jeho jednoduchá syntaxe. Každá volba (s výjimkou nejvýše postavených voleb) má nějaké rodiče a případně potomky. Každá volba je viditelná pouze, pokud je povolen její rodič. Voleb je několik typů: bool, tristate, string, hex, int. Řádky v kconfig souboru začínají klíčovým slovem, za kterým může následovat několik argumentů. Každá konfigurační volba začíná řádkem, na kterém je *config NAZEV_VOLBY* a na dalších řádcích následují její atributy.

Na následující ukázce bude popsán základ syntaxe Kconfigu (ukázka pochází ze souboru jádra v *arch/arm/mach–tegra/Kconfig*.)

```
config MACH_HARMONY
       bool "Harmony board"
       depends on ARCH_TEGRA_2x_SOC
       select MACH_HAS_SND_SOC_TEGRA_WM8903 if SND_SOC
       help
         Support for nVidia Harmony development platform
```

Zdrojový kód 1 – ukázka syntaxe Kconfig souborů

config MACH_VENTANA

Udává, že se jedná o konfigurační volbu s názvem *MACH_HARMONY*, ve vygenerovaném konfiguračním souboru bude tato volba pojmenována *CONFIG_MACH_HARMONY* a pod stejným názvem je tento symbol identifikován v Makefilech a zdrojových souborech. Díky tomu lze například určitou část kódu zkompilovat pouze, pokud je symbol nastaven na ano, viz Zdrojový kód 2 – zdrojový kód podmíněný nastavením Kconfig symbolu.

```
#if defined(CONFIG_MACH_HARMONY)
/* Harmony specific code here */
#endif //CONFIG_MACH_HARMONY
```

Zdrojový kód 2 – zdrojový kód podmíněný nastavením Kconfig symbolu

bool "Harmony board"

Tento řádek udává 2 informace. První z nich že se jedná o typ volby bool, tedy o volbu s možnostmi pouze ano nebo ne. Druhá část udává, pod jakým názvem bude volba zobrazována při konfiguraci pomocí menuconfigu apod.

depends on ARCH_TEGRA_2x_SOC

Tímto je zajištěno, že tato volba nelze vybrat, pokud není zatržena nadřízená volba, což je v tomto případě *ARCH_TEGRA_2x_SOC*.

select MACH_HAS_SND_SOC_TEGRA_WM8903 if SND_SOC

Tímto je zajištěno, že v případě, že uživatel povolí tuto volbu a zároveň i volbu *SND_SOC*, automaticky se povolí i odpovídající symbol *MACH_HAS_SND_SOC_TEGRA_WM8903*.

help

Udává, že na následujícím řádku je popsána nápověda k dané volbě.

4.2 Ladění a kontrola

4.2.1 ldd

Program ldd vypisuje sdílené knihovny, na kterých předložená sdílená knihovna nebo spustitelný soubor závisí. Díky tomu lze snadno zjistit chybějící závislosti, případně že program využívá nesprávnou sdílenou knihovnu.

```
$ ldd /bin/bash
        linux-vdso.so.1 =>  (0x00007fff074e7000)
        libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007fac0e556000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fac0e352000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fac0df92000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fac0e796000)
```

Zdrojový kód 3 – ukázka použití ldd

4.2.2 gdb

GDB(GNU Debugger) je v linuxovém světě nejpoužívanější debugger. Jeho nevýhodou je poměrně obtížné ovládání, proto se často používá prostřednictvím nějakého frontendu(DDD, plugin v Eclipse apod.) Pokud však prostřednictvím něj spustíme program, který je na cílové platformě nestabilní, je možné pomocí něj zjistit, v čem je chyba. Spuštění programu pod GDB se provádí následujícím způsobem.

```
$ gdb --args echo "Hello GDB"
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
```

```
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>...
Reading symbols from /bin/echo...(no debugging symbols found)...done.
(gdb) run
Starting program: /bin/echo Hello\ GDB
Hello GDB
[Inferior 1 (process 16474) exited normally]
(gdb
```

Zdrojový kód 4 – ukázka použití GDB

4.2.3 strace

Strace je linuxový nástroj pro diagnostiku a ladění. Jeho úkolem je zachytávat a vypisovat systémová volání, která provádí volaný program. Pro každé systémové volání vypisuje jeho název, argumenty se kterými je voláno i návratovou hodnotou. Analýzou těchto volání lze detekovat a případně vyřešit řadu chyb i v programech bez dostupného zdrojového kódu. Mezi nejsnadněji detekovatelné chyby patří selhání při otevírání souboru nebo nenalezené sdílené knihovny. Na ukázce Zdrojový kód 5 – ukázka běhu strace je nástroj strace aplikován na příkaz *echo "strace demonstration"*.

```
$ strace echo "strace demonstration"
execve("/bin/echo", ["echo", "strace demonstration"], [/* 10 vars */]) = 0
brk(0)                                  = 0x1200000
uname({sys="Linux", node="tqm", ...})   = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76fcb000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = -1 ENOENT (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
read(3, "\177ELF\1\1\1\0\0\0\0\0\0\0\0\0\3\0(\0\1\0\0\0\210z\1\0004\0\0\0"..., 512) =
512
fstat64(3, {st_mode=S_IFREG|0755, st_size=1176952, ...}) = 0
mmap2(NULL, 1213696, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x76e7e000
mprotect(0x76f9a000, 28672, PROT_NONE)  = 0
mmap2(0x76fa1000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x11b) = 0x76fa1000
mmap2(0x76fa4000, 9472, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1,
0) = 0x76fa4000
close(3)                                = 0
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x76fca000
set_tls(0x76fca4c0, 0x76fcab98, 0x76fca4c0, 0x76fcd048, 0x76fcd048) = 0
mprotect(0x76fa1000, 8192, PROT_READ)   = 0
```

```
mprotect(0x76fcc000, 4096, PROT_READ)   = 0
getuid32()                              = 0
brk(0)                                  = 0x1200000
brk(0x1221000)                          = 0x1221000
write(1, "strace demonstration\n", 21strace demonstration
)  = 21
exit_group(0)                           = ?
+++ exited with 0 +++
```

Zdrojový kód 5 – ukázka běhu strace

Jak je na ukázce vidět, i jednoduché příkazy způsobí velké množství systémových volání. Pro detekování chyb v nestabilním programu jsou však zpravidla klíčové poslední volání před pádem aplikace, proto není nutné číst celé výpisy.

5 NÁSTROJE PRO TVORBU SYSTÉMU

Vzhledem k množství a rychlosti tvorby nových embedded systémů vznikly nástroje, které celou činnost značně usnadňují a integrují v sobě většinu požadovaných nástrojů. K dispozici je poměrně veliké množství těchto nástrojů, proto jsou dále popsány pouze ty, které byly v rámci našeho projektu vyzkoušeny.

5.1 Buildroot

Buildroot je sada Makefilů a patchů, pomocí kterých je možné vygenerovat křížový překladač, zavaděč, linuxové jádro a root filesystém. [11]

Pro konfiguraci buildrootu se používá jazyk Kconfig, který je popsán v samostatné kapitole. Konfigurace Buildrootu je možná pomocí rozhraní pro CLI nebo GUI. Samotný Buildroot v sobě obsahuje velké množství balíčků a je poměrně jednoduché přidat další.

5.1.1 Stažení nástroje Buildroot

Buildroot je možné stáhnout v několika verzích. Přibližně každé 3 měsíce vychází stabilní verze. Pro ty, co vyžadují co možná nejnovější verze jsou k dispozici denní snapshoty nebo přímý přístup do Git repozitáře.

5.2 PTXdist

Stejně jako projekt Buildroot je i projekt PTXdist zaměřený na vygenerování kompletních systémů pro vestavěné zařízení. A podobně se i jeho struktura skládá z velké části z Makefilů a Kconfig souboru. Oproti Buildrootu ale PTXdist působí výrazně přehledněji a předvídatelněji. Jednotlivé sestavené části lze libovolně vyřazovat a překompilovávat, instalaci jednotlivých balíků lze provést po krocích a snadněji tak analyzovat potencionální chyby. Navíc za vývojem nástroje PTXdist stojí firma Pengutronix, která se zabývá zakázkovým portováním Linuxu na různé platformy a hostuje i projekt Barebox. [12]

Součástí je i nástroj pro generování kostry pro různé druhy nových balíčků, který výrazně usnadňuje přidávání nového softwaru.

5.2.1 Části nástroje PTXdist

1) ptxdist program

Program ptxdist je nainstalován na vývojovém počítači, spouští se pro vyvolání všech akcí, jako je zkompilování nějakého balíčku, vygenerování obrazů, jádra apod. Obyčejně se program ptxdist spouští ve workspace adresáři, který obsahuje všechny potřebné soubory ke správnému běhu.

2) *Konfigurační systém*

Konfigurační systém se používá pro upravení konfigurace, která obsahuje informace o tom, která balíčky se mají sestavit a jaké jsou nastavení projektu.

3) *Patche*

PTXdist obsahuje mechanismus pro automatické aplikování patchů na různé balíčky. To je nutné z toho důvodu, že řada balíčků obsahuje chyby – především v ohledu na křížové překládání a je tak nutné na ně aplikovat v rámci PTXdist opravy.

4) Popis balíčků

Pro každou softwarovou komponentu PTXdist obsahuje „recept“ – soubor příkazů a akcí, který je nutný pro získání, sestavení a nainstalování dané komponenty do systému. Každý balíček má navíc vlastní soubor pro konfigurační systém.

5) Toolchainy

PTXdist neobsahuje žádný předkompilovaný toolchain, ale je schopný si sestavit toolchainy, které jsou k dispozici v rámci projektu OSELAS.Toolchain().

6) Balíček pro podporu desky

Board Support Package(BSP) – Předkonfigurovaný balíček pro podporu konkrétní desky. Zpravidla se dodává k nějakému hardwaru, ale existují i obecné BSP připravené pro přizpůsobení na míru konkrétní platformě.

[13]

5.2.2 Stážení a zkompilování nástroje PTXdist

Následující seznam linuxových příkazů je možné použít ke zkompilování nástroje PTXdist. Program si v případě nesplněných závislostí může vyžádat instalaci dalších balíčků. Alternativou k uvedenému postupu je použít verzi dostupnou v balíčkovacím systému použité distribuce.

```
$ wget http://www.ptxdist.org/software/ptxdist/download/ptxdist-2013.01.90.tar.bz2
$ tar xzf ptxdist-2013.01.90.tar.bz2
$ cd ptxdist-2013.01.90
$ ./configure
$ make
$ sudo make install
```

Zdrojový kód 6 – stáhnutí a kompilace PTXdist

5.3 OpenEmbedded

Kromě již zmíněných nástrojů existuje i framework OpenEmbedded, za kterého dále vychází Yocto Project a Embeddded Linux Development Kit (ELDK). Tento nástroj již nevyužívá Makefily, ale namísto toho je složený z takzvaných BitBake receptů, které jsou odvozené od systému, jaký používá linuxová distribuce Gentoo.

Cílem OpenEmbedded a odvozených je spíše vytvoření distribuce s balíčkovacím systémem než vygenerování jednoho systému. Pro velkou část projektů je tento postup zbytečně komplikovaný a nepřináší užitek.

6 VYTVOŘENÍ PODPORY PRO NOVOU DESKU

6.1 Podpora v zavaděči

Nejlepší postup pro vytvoření podpory zavaděče pro novou desku je vyjít z již existující podpory pro podobnou desku. Zavaděče však trpí několika problémy, z nichž asi nejvýraznějším je nedostatečná nebo nekorektní dokumentace.

6.1.1 U-boot

V případě U-bootu není přes velmi rozsáhlou dokumentaci týkající se běhu U-bootu žádná oficiální dokumentace týkající se portování U-bootu na nové desky, i když lze nalézt informace z různých dalších zdrojů. Tyto postupy však jsou jen na určitou verzi U-bootu a mohou být zastaralé. Mezi použitelné zdroje lze zařadit následující: [14], [15], [16], [17].

Následující seznam kroků je možné použít jako výchozí cestu pro vytvoření podpory U-bootu pro novou desku. Některé desky však mohou vyžadovat přidání dalších ovladačů, nebo další přizpůsobení.

1. Stažení vhodné verze U-bootu – oficiální stabilní verze, větev pro danou platformu, nebo verze upravená od výrobce dané desky
2. Nalezení co nejpodobnější již podporované desky
3. Vytvoření konfiguračního souboru v *include/configs/jméno_desky.h* na základě již existujícího konfiguračního souboru pro jinou desku
4. Vytvoření podpůrného kódu v adresáři *board/jméno_desky/*
5. Přidání desky do souboru MAKEALL – v nových verzích U-bootu již není potřeba
6. Přidání desky do souboru Makefile, opět podobně jako u vzorové desky
7. Zkontrolovat, jestli je definovaný odpovídající MACH_TYPE v souboru include/asm-arm/mach-types.h, pokud ne, je nutné ho přidat.

[14]

6.1.2 Barebox

Barebox má v dokumentaci sekci, která se zabývá portováním na novou desku, ale není úplně kompletní a některé postupy – například pojmenování souborů – nejsou

v existujících zdrojových kódech dodržovány, což zhoršuje orientaci. I pro Barebox je možné nalézt další zdroje týkající se jeho portování, například [18], [19], [20].

Pro portování Bareboxu je vhodné využít obdobný postup, jako pro portování U-bootu, tedy vyjít z podpory pro nějakou podobnou desku a pouze ji přizpůsobit. Navíc je možné využít jako základ kód z U-bootu, který obsahuje podporu pro větší množství desek.

6.2 Podpora v linuxovém jádře

Náročnost vytváření podpory pro novou desku silně závisí na verzi použitého jádra a subarchitektury. Ve starších jádrech bez podpory Device Tree je nutné pro podporu nové desky vytvořit jeden nebo více souborů se zdrojovým kódem, případně formou podmíněné kompilace poupravit již existující soubory.

Protože je nyní doporučování používat Device Tree, je dále popisován pouze tento způsob.

6.2.1 Struktura Device Tree Source souborů

Každý DTS soubor je uspořádán do stromové struktury, který je tvořena Device Tree uzly (Nodes). Až na výjimky každý uzel popisuje nějaké zařízení nebo sběrnici a obsahuje vlastnosti, které jsou zapisovány stylem název = hodnota. Jako příklad je uvedená zkrácená ukázka pro podporu tlačítkové klávesnice připojené na GPIO piny procesoru.

```
/* Uzel popisující GPIO klávesnici s názvem gpio-keys */
gpio-keys {
                /* Klíč compatible udává, jaký typ zařízení uzel *
                 * popisuje a jaký ovladač se pro něj použije    */
                compatible = "gpio-keys";

                /* Uzel popisující jednu klávesu s názvem function-1  */
                function-1 {
                        /* Popisek klávesy */
                        label = "Function 1";
                         /* Definice GPIO pinu, na kterém *
                          * je tlačítko připojeno            */
                        gpios = <&gpio3 21 1>;
                         /* Kód klávesy, který se při stisknutí *
                          * tlačítka v systému objeví           *
```

```
                        * (zde je to F1)                        */
                       linux,code = <59>; /* KEY_FN_1 */
               };

               /* Uzel popisující jednu klávesu s názvem function-2  */
               function-2 {
                       label = "Function 2";
                       gpios = <&gpio2 27 4>;
                       linux,code = <60>; /* KEY_FN_2 */
               };
};
```

Zdrojový kód 7 – ukázka Device Tree uzlu

7 VYTVOŘENÍ SYSTÉMU

Nástroje pro tvorbu systému již nemusejí obsahovat žádné platformě specifické věci, které by bylo nutné přizpůsobovat a stačí je k vygenerování funkčního systému správně nastavit.

7.1 Buildroot

Nejprve je nutné mít stažený vybraný archív s Buildrootem - informace kde stáhnout tento archív je v kapitole 5.1.1 Stažení nástroje Buildroot. Po rozbalení tohoto archívu do adresáře, ve kterém chceme vytvářet systém již postačuje z adresáře s buildrootem zavolat příkaz *make menuconfig*, pomocí kterého buildroot nakonfigurujeme přesně podle požadavků. Nakonfigurovaný buildroot poté spustíme příkazem *make*. Jednoduchost Buildrootu má však svoje úskalí – pokud proces vyžaduje použití vlastních konfiguračních souborů, vlastního upraveného jádra a podobně, stává se aktuální konfigurace nepřenositelná na jiný počítač. Řešením je používat namísto pevných cest k souborům cesty, které jsou relativní k adresáři Buildrootu, nebo vytvořit skript s definicí těchto cest do proměnných prostředí a volat Buildroot z něj. Ukázku takového skriptu je možné nalézt v příloze P III: Skript pro práci s Buildrootem.

7.2 PTXdist

Dále uvedený postup předpokládá, že PTXdist už je v systému nainstalovaný buď pomocí postupu popsaného v popsaný v kapitole 5.2.2 nebo pomocí balíčkovacího systému.

Před započetím práce je nutné mít nainstalovaný alespoň jeden toolchain, který bude použit ke zkompilování a sestavení všech nutných součástí. Pro PTXdist se doporučuje použít toolchain sestavený pomocí OSELAS.Toolchain(), což je systém pro vygenerování toolchainu využívající PTXdist. Toolchainy je možné získat několika způsoby. Tím nejjednodušším je nainstalovat hotový toolchain pomocí balíčkovacího systému – například pro Ubuntu/Debian existuje repozitář, který stačí přidat mezi zdroje balíčkovacího systému a poté z něj nainstalovat vybraný toolchain.[21] Alternativně lze stáhnout některé již sestavené toolchainy, které distribuují někteří výrobci desek.

V případě, že pro použitou platformu není dostupný žádný hotový toolchain, nepoužíváme distribuci založenou na Debianu nebo požadujeme některé specifické věci, které generický toolchain neposkytuje, je nutné si sestavit vlastní toolchain.

7.2.1 Sestavení OSELAS.Toolchain()

Prvním krokem je stažení archívu obsahující PTXdist konfiguraci pro sestavení toolchainu ze stránek PTXdist zde: *http://www.ptxdist.de/oselas/toolchain/download/*. Následně je nutné archív rozbalit, přejít do rozbaleného adresáře, vybrat požadovaný toolchain a spustit kompilaci.

Pro nejaktuálnější verzi OSELAL Toolchainu (2012.12.1) a procesor s jádrem Cortex A8 by postup vypadal následovně:

```
$ tar xf OSELAS.Toolchain-2012.12.0.tar.bz2

$ cd OSELAS.Toolchain-2012.12.0

$ ptxdist select ptxconfigs/arm-cortexa8-linux-gnueabi_gcc-4.7.3_glibc-2.16.0_binutils-
2.22_kernel-3.6-sanitized.ptxconfig

$ ptxdist go
```

Zdrojový kód 8 – sestavení OSELAS toolchainu

Zkompilovaný toolchain se nainstaluje do adresáře */opt/OSELAS.Toolchain-2012.12.0/arm-cortexa8-linux-gnueabi/gcc-4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/*. Před prvním použitím je ještě doporučeno k vygenerovanému toolchainu nastavit práva pouze pro čtení, aby se zabránilo jeho poškození.

7.2.2 Vytvoření a přizpůsobení BSP

V terminologii emebedded systémů se kolekce nástrojů a programů pro specifickou desku nazývá BSP (Board Support Package) neboli balík pro podporu desky. Tento termín nemá žádný zavedený překlad, proto se dále využívá zkratka BSP.

V rámci PTXdist se BSP skládá z konfiguračních souborů, patchů, specifických balíků a podobně. Na webu PTXdist lze stáhnout několik hotových BSP, další jsou k dispozici například od některých výrobců evaluation kitů. Jako příklad bude použit Generic BSP, který je možné stáhnout na adrese *http://oselas.com/oselas/bsp/pengutronix/download*. Po extrahování a přejití do extrahovaného adresáře již je možné spouštět příkazy PTXdistu, pomocí kterých se projekt nakonfiguruje. Následující příklad ilustruje přípravu Generic BSP.

```
$ wget http://oselas.com/oselas/bsp/pengutronix/download/OSELAS.BSP-Pengutronix-Generic-2012.12.0.tgz

$ tar xf OSELAS.BSP-Pengutronix-Generic-2012.12.0.tgz

$ cd OSELAS.BSP-Pengutronix-Generic-2012.12.0.tgz
```

Zdrojový kód 9 – získání BSP

7.2.2.1 Adresářová struktura BSP v PTXDistu

Adresář	**Popis**
configs	Obsahuje konfigurační soubory ptxdistu a jednotlivých balíčků (například jádra, zavaděče, apod.) V podadresáři *configs/platform-název_platformy/patches* se obvykle umisťují patche jádra a zavaděče.
local_src	Obsahuje archívy nebo adresáře pro balíčky specifické k projektu – například vlastní aplikace v projektu.
patches	Obsahuje patche jednotlivých balíčků konfigurovaných přes *menuconfig*.
platform-název_platformy	V tomto adresáři probíhá kompilace všech součástí a sestavování cílového obrazu. Sestavené obrazy je možné nalézt v podadresáři *images*.
projectroot	Obsahuje konfigurační soubory kopírované do cílového systému. Například konfigurace sítě, skripty pro start aplikace po startu apod.
rules	Obsahuje konfigurační soubory s příponou .in, které ovládají zobrazování v menuconfigu a závislosti balíčku a soubory příponou .make, které popisují proces sestavování balíčku, udávají jaké výsledné soubory se kopírují do cílového systému a další.
src	Do této složky se stahují veškeré používané balíčky.
tests	Tato složky obsahuje automatizované testy, kterými je možné ověřit správnost sestaveného systému

Tabulka 4 – adresářová struktura BSP v PTXdistu

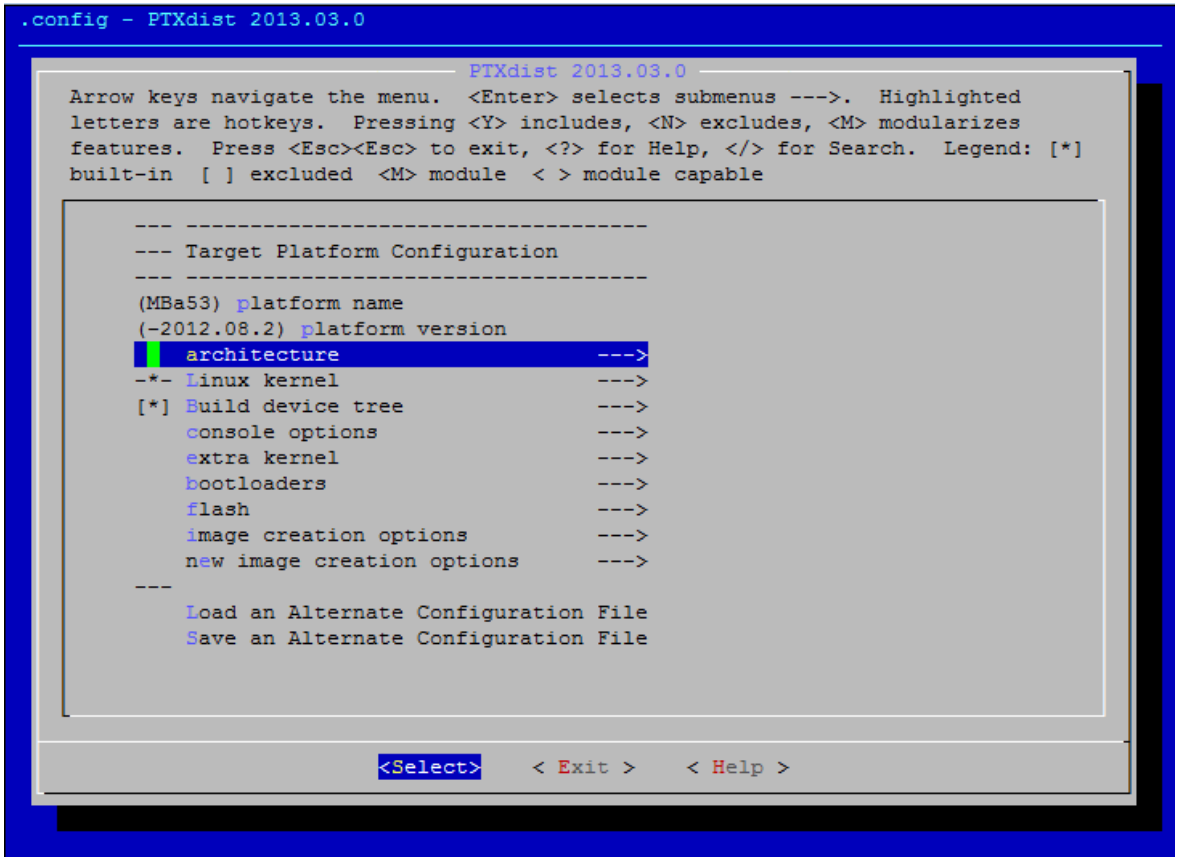
7.2.2.2 Konfigurace platformy

První věc, kterou je nutné udělat, je nastavit toolchain, který bude projekt využívat. To je možné učinit dvěma způsoby. Jedním z nich je použití příkazu *ptxdist toolchain /cesta/k/toolchainu*, nebo ho lze nastavit v dialogu vyvolaném příkazem *ptxdist platformconfig*.

```
$ ptxdist toolchain /opt/OSELAS.Toolchain-2011.11.1/arm-cortexa8-linux-gnueabi/gcc-
4.6.2-glibc-2.14.1-binutils-2.21.1a-kernel-2.6.39-sanitized/bin'
```

Zdrojový kód 10 – nastavení toolchainu pomocí příkazu ptxdist toolchain

Příkaz *ptxdist platformconfig* oproti tomu vyvolá dialog, který je možné vidět na Obrázek 8 – ptxdist platformconfig.



Obrázek 8 – ptxdist platformconfig

V tomto dialogu je možné nastavit toolchain v sekci architecture → toolchain. V sekci architecture lze dále nastavit řadu důležitých voleb, jako je cílová architektura a různá nastavení zvyšující bezpečnost. V dalších sekcích lze pak ovlivnit, jestli se bude v rámci systému sestavovat linuxové jádro, jaký bude použit zavaděč, generované obrazy a další.

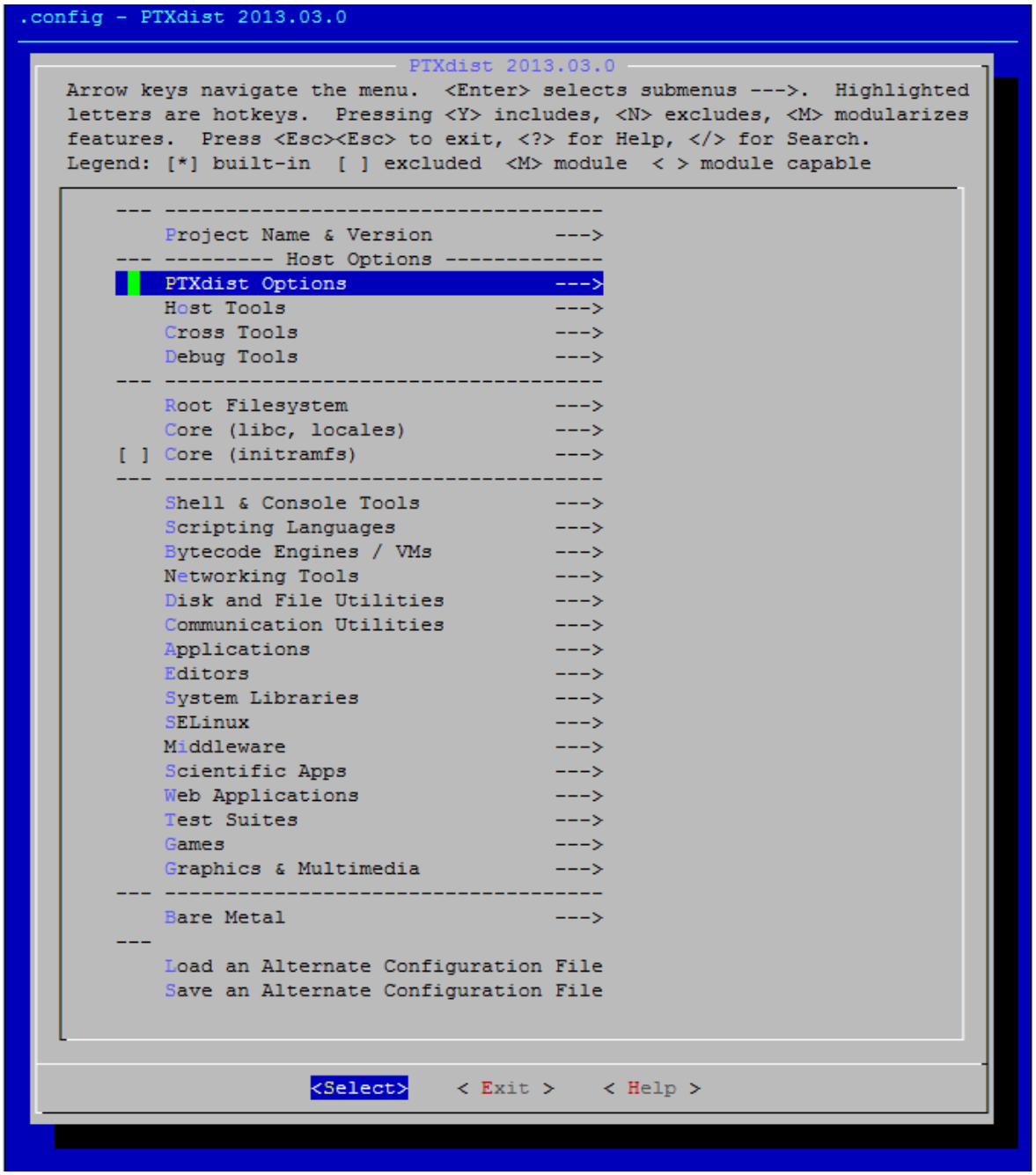
7.2.2.3 Konfigurace jádra a softwaru

V rámci PTXdist je možná nakonfigurovat jádro příkazem *ptxdist kernelconfig*, který zobrazí to samé jako *make menuconfig* v adresáři jádra. Před spuštěním příkazu musí být korektně nastavená platforma, jinak by se v kernelconfigu mohly objevit nastavení pro jinou procesorovou architekturu, nebo by se vůbec nespustila – například v případě špatně nastaveného toolchainu.

V konfiguraci jádra je nutné vybrat správný procesor, desku, ovladače a jiné nastavení, která jsou specifická pro použitou desku. Vybrané nastavení by také měly odrážet požadovanou aplikaci a neměly by být vybrány nepotřebné součásti. Také je výhodnější všechny součásti kompilovat jako součásti jádra a ne jako moduly, protože zavádění modulů dynamicky sebou nese zbytečný overhead.

Konfigurace softwaru se provádí příkazem *ptxdist menuconfig*. V zásadě zde platí obdobné zásady jako u konfigurace jádra, tedy že by měly být vybrány jen nutné součásti. Množství instalovaného softwaru sice nemá vliv na dobu bootu, ale zbytečně prodlužuje dobu kompilace systému a může zanést zbytečné bezpečnostní rizika.

PTXdist obsahuje velké množství aplikací, z nichž některé jsou přizpůsobeny pro křížové překládání na jiné platformy. Kromě všech možných linuxových nástrojů a aplikací je zde možné nalézt i multimediální aplikace, knihovny Qt & GTK, nebo třeba Java Runtime Environment a množství balíků stoupá s každou verzí. V případě, že nějaký software chybí, je také poměrně jednoduché vytvořit vlastní balík a integrovat ho.



Obrázek 9 – volby softwaru v PTXdist

7.2.3 Vytvoření vlastních balíčků

I přes veliké množství software v PTXdistu je často nutné přidat vlastní software. Je sice možné vlastní programy samostatně přeložit a poté nakopírovat do výsledných obrazů, ale zpravidla je výrazně jednodušší vytvořit pro vlastní software balíčky, které zajistí kompilaci a kopírování do cílového systému samy. V PTXdistu jsou pro vytváření vlastních balíčků šablony a průvodce, který celý proces usnadňuje.

7.2.3.1 Použití průvodce

Pro vytvoření nového balíčku pomocí průvodce slouží příkaz *ptxdist newpackage typ_balíčku*. Pokud je tento příkaz spuštěn bez zadaného typu balíčku, vypíše možnosti, jaké typy balíčků jsou k dispozici, viz Zdrojový kód 11 – použití ptxdist newpackage bez zadání typu balíčku.

```
$ ptxdist newpackage

usage: 'ptxdist newpackage <type>', where type is:

  host                      create package for development host
  target                    create package for embedded target
  cross                     create cross development package
  klibc                     create package for initramfs built against klibc

  src-autoconf-lib          create autotoolized library
  src-autoconf-prog         create autotoolized binary
  src-autoconf-proglib      create autotoolized binary+library
  src-cmake-prog            create cmake binary
  src-qmake-prog            create qmake binary
  src-linux-driver          create a linux kernel driver
  src-make-prog             create a plain makefile binary
  src-stellaris             create stellaris firmware
  font                      create a font package
  file                      create package to install existing files
  kernel                    create package for an extra kernel
  barebox                   create package for an extra barebox
  image-tgz                 create package for a tgz image
  image-genimage            create package for a genimage image
```

Zdrojový kód 11 – použití ptxdist newpackage bez zadání typu balíčku

Kompletní popis k čemu který je balíček lze nalézt v PTXdist manuálu [22].

Příklad ukázaný v Zdrojový kód 12 demonstruje vytvoření balíčku pro vlastní aplikaci postavenou na frameworku Qt s použitím qmake.

```
$ ptxdist newpackage src-qmake-prog


ptxdist: creating a new 'src-qmake-prog' package:


ptxdist: enter package name...........: MyQtDemoApp
ptxdist: enter version number.........: 1.00
ptxdist: enter package author.........: Roman Dosek
ptxdist: enter package section........: project_specific


generating rules/MyQtDemoApp.make
generating rules/MyQtDemoApp.in


local_src/myqtdemoapp-1.00 does not exist, create? [Y/n] Y
./
./wizard.sh
./install.pri
./@name@.cpp
./@name@.pro
```

Zdrojový kód 12 – ukázka vytvoření balíčku s pomocí průvodce

7.2.3.2 Úprava .in a .make souborů

Pro některý software jsou balíčky vytvořené pomocí průvodce již kompletně funkční a není nutné je dále upravovat, jindy je ale nutné balíčky vytvořené pomocí průvodce upravovat, například kvůli kopírování dalších souborů do cílového systému.

7.2.3.3 Testování balíčku

Sestavení každého balíčku je rozděleno do několika částí, a tyto části lze provádět samostatně a ověřit tak každý krok při vytváření nového balíčku.

1) Stažení

Příkaz: *ptxdist get název_balíčku*

Popis: Zdroj, odkud se balíček stahuje je definován v příslušném make souboru. Zdrojem může být http, ftp, git, nebo třeba lokální soubor (file).

2) Extrahování

Příkaz: *ptxdist extract název_balíčku*

Popis: Extrahování balíčku v závislosti na příponě, jako umá soubor definovanou v make souboru.

3) Příprava

Příkaz: *ptxdist prepare název_balíčku*

Popis: V této fázi se spouští zpravidla configure skript pro přípravu balíčku ke kompilaci, ale v závislosti na typu balíčku je možné zde spouštět libovolné příkazy.

4) Kompilace

Příkaz: *ptxdist compile název_balíčku*

Popis: Ke zkompilování balíčku se zpravidla používá příkaz make. PTXdist obsahuje i několik šablon pro nejčastěji používané kompilační systémy (Autotools, qmake, CMake,…)

5) Instalace

Příkaz: *ptxdist install název_balíčku*

Popis: Nainstaluje potřebné soubory do hostitelského systému – například knihovny, na kterých závisí další programy. Adresář, do kterého se soubory kopírují je *sysroot*.

6) Cílová instalace

Příkaz: *ptxdist targetinstall název_balíčku*

Popis: Nainstaluje soubory určené pro cílový systém do adresáře *root*.

7.2.4 Kompilace a vytvoření obrazů

Pokud je v PTXdistu vše správně nastaveno, je možné přejít k samotné kompilaci a vytvoření obrazů. Pro kompilaci slouží příkaz *ptxdist go*. Ten zajistí kompilaci všech balíčků, které ještě nejsou zkompilovány. V případě, že se během běhu *ptxdist go* nevyskytly žádné chyby, lze vytvořit výsledné obrazy příkazem *ptxdist images*. Jaké typy obrazů budou vytvořeny a v jakých formátech závisí na výběru, který je možné učinit v konfiguraci platformy.

8 SOFTWAROVÉ ŘEŠENÍ NA CÍLOVÉ PLATFORMĚ

Tato kapitola shrnuje software instalovaný na cílové platformě a jeho použití.

8.1 Init systém

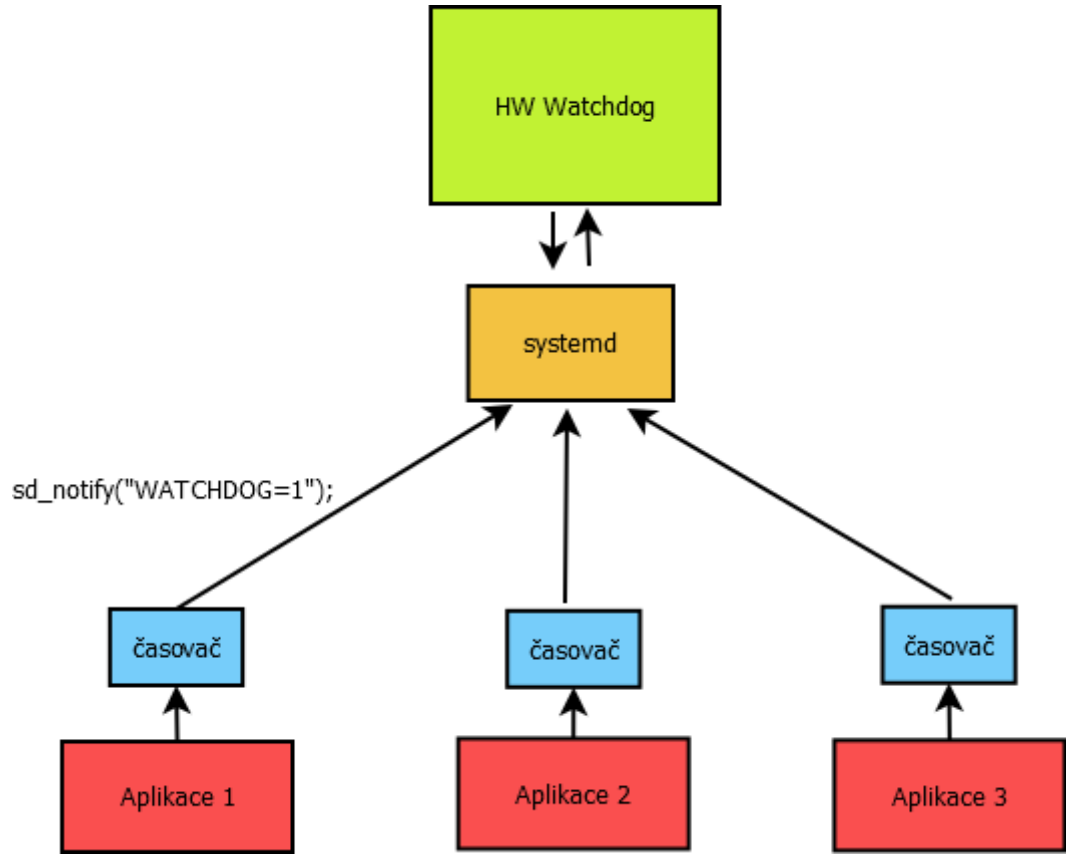
Jako init systém byl použit *systemd*, který je poměrně velkou novinkou. Důvodů k jeho použití bylo hned několik. V první řadě není založen na shellových skriptech jako starší systémy *System V init* nebo *BSD init*, ale používá deklarativně zapsané soubory, ve kterých jsou jasně zapsané zavislosti jednotlivých služeb. Mimo zrychlení způsobené tím, že není nutné interpretovat shell skripty, je tak nyní s použitím tohoto init systému celý bootovací proces možné paralelizovat bez obav, že by docházelo k uváznutí procesů.

Systemd provádí sledování procesů a umožňuje nadefinovat akce, které se mají provést v případě ukončení procesů. Služby je navíc možné spouštět prostřednictvím Socketů nebo D-Busu.

8.2 Watchdog

Použitý modul TQMa53 obsahuje hardwarový watchdog, díky kterému je možné modul zrestartovat v případě potíží. Nejjednodušší použití watchdogu spočívá v tom, že po jeho zapnutí je nutné v určitých časových intervalech zapisovat do specifického souboru. Protože je ale v rámci aplikace nutné sledovat více částí, byl zvolen postup, který zahrnuje systemd jako prostředníka pro použití watchdogu.

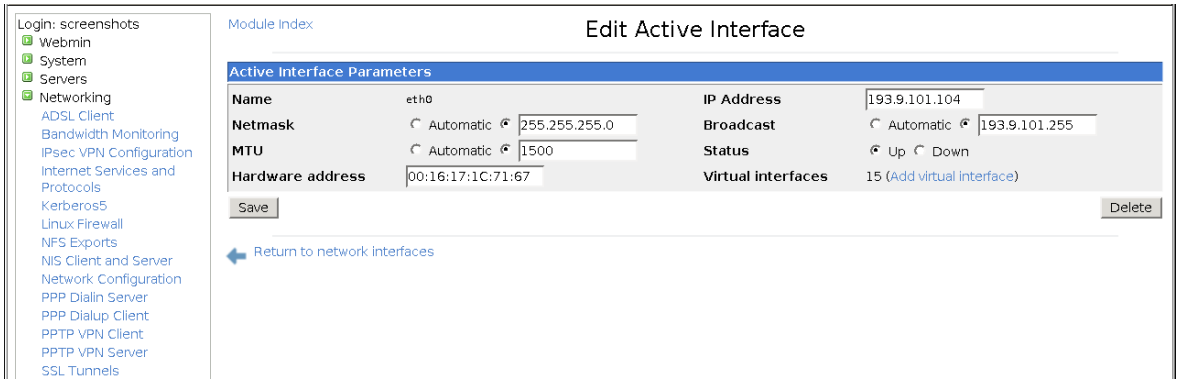
1. Hardwarový watchdog je v pravidelných intervalech notifikován pomocí systemd. V případě zaseknutí systemd provede systém hard reset.
2. Aplikace v pravidelných intervalech notifikuje systemd. Pokud aplikace neodešle notifikaci, systemd se ji pokusí zrestartovat a pokud se to v určitém počtu pokusů nebo čase nepodaří, restartuje systém.



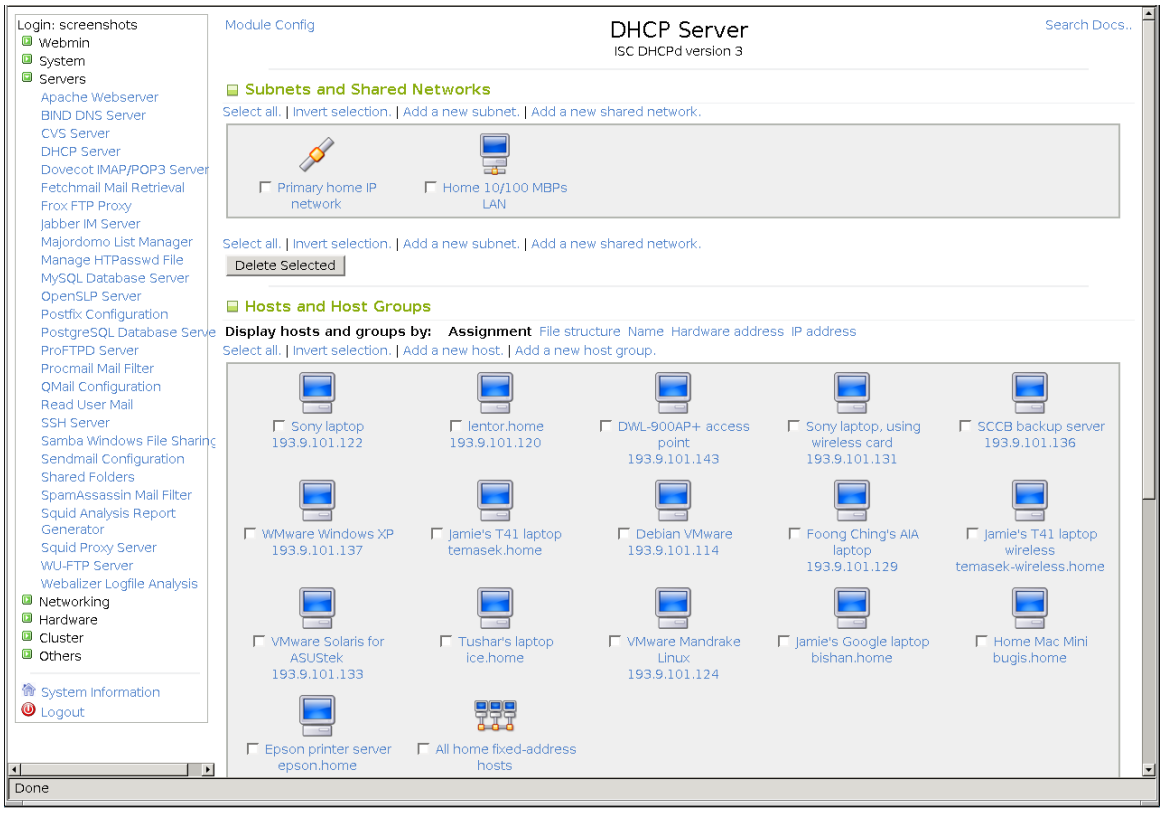
Obrázek 10 – struktura použití watchdogu

8.3 Web rozhraní pro správu systému

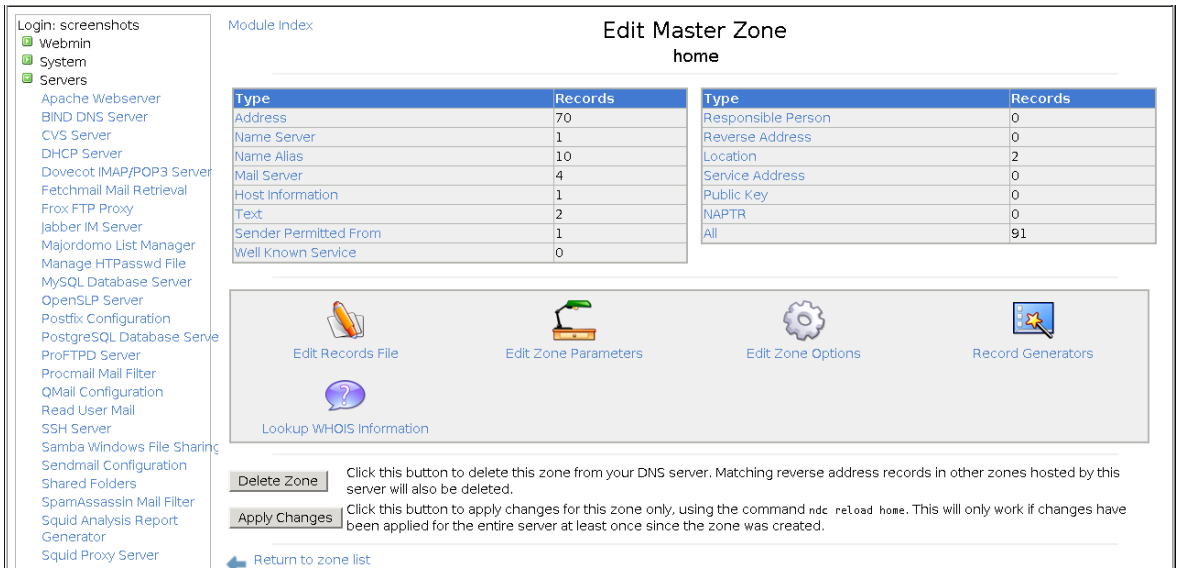
Web rozhraní bylo implementováno instalací open source systému webmin, který nabízí nepřeberné množství modulů pro správu celého systému. Na obrázcích níže je několik screenshotů modulů, které se používají pro správu sítě.



Obrázek 11 – webmin: konfigurace síťového rozhraní



Obrázek 12 – webmin: konfigurace DHCP serveru



Obrázek 13 – webmin: konfigurace DNS serveru

ZÁVĚR

Při implementaci operačního systému Linux na platformě i.MX53 (ARM Cortex A8) jsme získali několik nových poznatků.

Jako zavaděč se u tohoto CPU více osvědčil Barebox, který sice obsahuje podporu menšího množství hardwaru, ale způsob práce s ním více připomíná práci s plnohodnotným linuxovým systémem. Za jednu z jeho největších výhod oproti U-bootu lze také považovat fakt, že skripty lze ukládat do souborů a díky tomu se snadněji editují, udržují i používají a lze na první pohled rozlišit mezi standardním příkazem zavaděče a uživatelským skriptem.

Z nástrojů na vytvoření systému se subjektivně lépe pracovalo s nástrojem PTXdist. Nástroj Buildroot je naproti tomu velice vhodný pro vytvoření jednoduchého systému pro cílovou desku, ale je nutné počítat s tím, že vytvořená konfigurace je obtížněji přenositelná na jiný počítač. PTXdist umožňuje větší nastavení a je lépe připraven na práci v týmu a úpravy balíčků, daní za to mírně větší komplexita nástroje.

Výsledkem práce je reálně nasaditelný systém s moderním procesorem Freescale i.MX53, u nějž výrobce garantuje dostupnost až do roku 2020.

SEZNAM POUŽITÉ LITERATURY

[1] VENKATESWARAN, Sreekrishnan. Essential Linux device drivers. Upper Saddle River: Prentice Hall, c2008, xxx, 714 s. ISBN 978-0-132-39655-4.

[2] CORBET, Jonathan. Linux device drivers. 3rd ed. Sebastopol: O´Reilly, 2005, xviii, 615 s. ISBN 05-960-0590-3.

[3] LOVE, Robert. Linux kernel development. 3rd ed. Upper Saddle River: Addison-Wesley, c2010, xx, 332 s. ISBN 06-723-2946-8.

[4] GRÖTKER, Thorsten. The developer's guide to debugging. 2nd ed. North Charleston, S.C.?: [CreateSpace Independent Publishing Platform], c2012, xx, 242 s. ISBN 978-1470185527.

[5] STALLMAN, Richard M a Roland MCGRATH. GNU make: a program for directing recompilation : GNU make version 3.79.1 : June, 2002. Boston: Free Software Foundation, 2002, vi, 184 s. ISBN 18-821-1482-5.

[6] Building embedded Linux systems. 2nd ed. Sebastopol: O´Reilly, 2008, xx, 439 s. ISBN 978-0-596-52968-0.

[7] QNX Neutrino RTOS. QNX [online]. ©2004–2013 [cit. 2013-04-02]. Dostupné z: http://www.qnx.com/products/neutrino–rtos/neutrino–rtos.html#overview

[8] The History of ARM Linux. The ARM Linux Project [online]. © 2013 [cit. 2013-04-02]. Dostupné z: http://www.arm.linux.org.uk/docs/history.php

[9] RUSLING, David. Kernel Upstreaming. In: Linaro Blog [online]. 2011 [cit. 2013-04-02]. Dostupné z: http://www.linaro.org/linaro-blog/2011/06/09/kernel-upstreaming/

[10] PETAZZONI, Thomas. FREE ELECTRONS. Linux kernel: consolidation in the ARM architecture support. 2012. Dostupné z: http://free-electrons.com/pub/conferences/2012/lsm/arm-kernel-consolidation/arm-kernel-consolidation.pdf

[11] Buildroot [online]. © 1999–2005 Erik Andersen, 2006-2012 The Buildroot developers [cit. 2013–04–02]. Dostupné z: http://buildroot.uclibc.org/

[12] PTXdist – Reproducible Embedded Linux Systems [online]. 2013 [cit. 2013-04-02]. Dostupné z: http://www.ptxdist.org/software/ptxdist/index_en.html

[13] Application Note – Installing PTXdist-2012.12.0. Hildesheim, 2012. Dostupné z: http://www.ptxdist.org/software/ptxdist/appnotes/AppNote_InstallingPtxdist.pdf

[14] OPDENACKER, Michael. FREE ELECTRONS. Porting U-boot. © 2004-2009. Dostupné z: http://free-electrons.com/doc/porting-u-boot.pdf

[15] HALLINAN, Christopher. Bootloaders in Embedded Linux Systems. In: Pearson Education, Informit [online]. 2010 [cit. 2013-04-02]. Dostupné z: http://www.informit.com/articles/article.aspx?p=1647051

[16] Porting U-Boot to a new board. STMICROELECTRONICS. STLinux [online]. © 2008-2013 [cit. 2013-04-02]. Dostupné z: http://www.stlinux.com/u-boot/porting

[17] RAGHUNANDAN, ES. Porting U-Boot to a New Board [online]. 2012 [cit. 2013-04-02]. Dostupné z: http://portinguboottoanewboard.blogspot.com

[18] Adapting a new Board. In: Barebox Developer's Manual [online]. 2011 [cit. 2013-04-02]. Dostupné z: http://barebox.org/documentation/barebox-2011.05.0/dev_board.html

[19] Barebox – Getting started. Elec4fun [online]. 2011 [cit. 2013-04-02]. Dostupné z: http://www.elec4fun.fr/2011-03-30-10-16-30/2011-03-31-13-08-45/startwithbarebox

[20] Barebox. In: Crash Course wiki [online]. 2012 [cit. 2013-04-02]. Dostupné z: http://www.crashcourse.ca/wiki/index.php/Barebox

[21] OSELAS®.Toolchain(). In: PTXdist [online]. 2013 [cit. 2013-04-02]. Dostupné z: http://www.ptxdist.de/oselas/toolchain/index_en.html

[22] PENGUTRONIX. How to become a PTXdist Guru. Hildesheim, 2012. Dostupné z: http://www.pengutronix.de/software/ptxdist/appnotes/OSELAS.BSP-Pengutronix-Generic-arm-Quickstart.pdf

[23] DONGWOOK, Kang. ETRI. Snapshot Booting on Embedded Linux. Daejeon, 2011. Dostupné z: http://elinux.org/images/c/c3/Elc2011_kang.pdf

[24] Suspend to Disk for ARM. In: Embedded Linux Wiki [online]. 2011 [cit. 2013-05-08]. Dostupné z: http://elinux.org/Suspend_To_Disk_For_ARM

[25] QuickBoot. Qbiquitous [online]. © 2013 [cit. 2013-05-08]. Dostupné z: http://www.ubiquitous.co.jp/En/products/middleware/quickboot/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API Application Programming Interface

ARM Advanced RISC Machine

BSP Board Support Package

CAN Controller Area Network

CLI Command Line Interface

DTB Device Tree Blob

DTC Device Tree Compiler

DTS Device Tree Source

ELDK Embedded Linux Developmnet Kit

FDT Flattened Device Tree

GCC GNU Compiler Collection

GDB GNU Debugger

GPIO General Purpose Input/Output

LCD Liquid Crystal Display

LVDS Low-voltage differential signaling

OE OpenEmbedded

RTOS Real-Time Operating Systém

SoC System-on-Chip

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘÍKAZY U-BOOTU

Příkazy U–bootu jsou rozděleny do kategorií podle funkce a pro každou desku lze v konfiguračním souboru definovat, které z příkazů U–Bootu budou dostupné. Z těchto příkazů je navíc možné skládat další příkazy, které se poté spouští příkazem run.

Informační příkazy

bdinfo – vypíše obsah struktury Board Info

coninfo – vypíše dostupná konzolá zařízení a informace o

flinfo – vypíše informace o flash paměti

iminfo – vypíše hlavičku aplikačního obrazu

help – zobrazí nápovědu

Příkazy pro manipulaci s pamětí

base – vypsání nebo nastavení offsetu adresy

crc32 – výpočet kontrolního součtu

cmp – porovnání obsahu paměti

cp – kopírování obsahu paměti

md – zobrazení obsahu paměti

mm – změna obsahu paměti (automatická inkrementace)

mtest – jednoduchý test RAM paměti

mw – zápis do paměti (naplnění)

nm – změna obsahu paměti (konstantní adresa)

loop – nekonečná smyčka na rozsahu adres

Příkazy pro práci s Flash pamětí

erase – vymazání Flash paměti

protect – povolení nebo zakázání zámku pro zápis do Flash paměti

mtdparts – definování MTD oddílů kompatibilních s Linuxem

Příkazy pro spouštění programů

source – spuštění skriptu z paměti

bootm – spuštění aplikačního obrazu z paměti

go – spuštění aplikace z adresy

Příkazy pro práci se sítí a sériovou linkou

bootp – spuštění obrazu ze sítě s použitím protokolu BOOTP/TFTP

dhcp – spuštění DHCP klienta pro získání IP adresy/bootovacích parametrů

loadb – načtění binárního souboru přes sériovou linku (kermit)

loads – načtení S–Record souboru přes sériovou linku

rarpboot – spuštění obrazu ze sítě s použitím protokolu RARP/TFTP

tftpboot – spuštění obrazu ze sítě s použitím protokolu TFTP

Příkazy pro manipulaci s proměnnými prostředí

printenv – vypíše proměnné prostředí

saveenv – uloží proměnné prostředí do permanentní paměti

setenv – nastaví proměnno u prostředí

run – spustí příkaz z proměnné prostředí

bootd – spuštění výchozího obrazu

Příkazy pro práci s Flattened Device Tree

fdt addr – výběr FDT se kterým se bude pracovat

fdt list – výpis jedné úrovně

fdt print – rekurzivní výpis

fdt mknode – vytvoření nového uzlu

fdt set – nastavení vlastností uzlu

fdt rm – odstranění uzlu nebo jeho vlastností

fdt move – přesunutí FDT blobu na novou adresu

fdt chosen – opravení dynamických informací

Speciální příkazy

i2c – I2C subsystém

Ostatní příkazy

echo – výpis argumentů do konzole

reset – provedení resetu CPU

sleep – pozastavení běhu na určitou dobu

version – vypíše verzi monitoru

? – alias k příkazu 'help'

PŘÍLOHA P II: PŘÍKAZY BAREBOXU

Jelikož je Barebox forkem U-bootu, některé příkazy jsou shodné, nicméně Barebox se snaží chovat co nejvíc podobně Linuxu, proto je jako v Linuxu podobný systém adresářů a příkazy.

Standardní příkazy

clear – vyčištění obrazovky
cat – zobrazí obsah souboru
cd – změna aktuálního adresáře
mkdir – vytvoření adresáře
cp – zkopírování souboru
edit – otevře editor souboru
pwd – vypíše aktuální pracovní adresář
mount – připojení oddílu
ls – vypsání obsahu adreáře
rm – smazání souboru
rmdir – smazání adresáře
time – vypsání aktuálního času
umount – odpojení oddílu
global – vytvoření globální proměnné
export – exportuje proměnnou
false
exec – spuštění skriptu
insmod – načtení modulu
passwd – nastavení hesla k zavaděči
sh – spustí skript
true
test – příkaz pro porovnání hodnot
lsmod – výpis načtených modulů

Příkazy pro práci s GPIO a LED

gpio_direction_input – nastaví GPIO pin jako vstupní
gpio_direction_output – nastaví GPIO pin jako výstupní

gpio_get_value – získání hodnoty GPIO pinu

gpio_set_value – nastavení hodnoty GPIO pinu

led – nastavení hodnoty LED

trigger – přiřazení triggeru k LED

Informační příkazy

cpuinfo – zobrazí informace o CPU (specifické pro ARM)

devinfo – zobrazí informace o známých zařízeních a ovladačích

reginfo – zobrazí informace o některých registrech (specifické pro mpc5200)

iomem – zobrazí informace o I/O prostředcích a jejich využití

help – zobrazí nápovědu

Příkazy pro práci s UBI oddíly

ubiattach – připojení zařízení do UBI

ubimkvol – vytvoření UBI oddílu

ubirmvol – smazání UBI oddílu

Příkazy pro spouštění programů

source – spuštění skriptu z paměti

bootm – spuštění aplikačního obrazu z paměti

go – spuštění aplikace z adresy

linux16 – spuštění obrazu linuxu (pouze pro x86 platformu)

Příkazy pro manipulaci s pamětí

crc – výpočet kontrolního součtu

nand – manipulace s vadnými bloky v NAND paměti

protect – zakáže zápis do Flash paměti

unprotect – povolí zápis do flash paměti

md – zobrazí obsah paměti

erase – vymazání flash paměti

addpart – vytvoření oddílů na zařízení nebo souboru

delpart – odstranění oddílů na zařízení nebo souboru

memcmp – porovná obsah paměti

memcpy – zkopíruje obsah paměti

meminfo – zobrazí informace o paměti

mw – zapsání hodnoty do paměti nebo souboru

Příkazy pro manipulaci s proměnnými prostředí

printenv – vypíše proměnné prostředí

saveenv – uloží proměnné prostředí do permanentní paměti

setenv – nastaví proměnno u prostředí

loadenv – načtění proměnných prostředí z trvalého úložiště

magicvar – vypíše proměnné prostředí se speciálním významem a jejich význam

Příkazy pro práci se sítí a sériovou linkou

tftp – přenesení souboru přes tftp protokol

ping – ping síťového zařízení

nfs – načtení souboru z NFS serveru

dfu – spustí update firmwaru s pomocí DFU (Device Firmware Update) protokolu

Ostatní příkazy

echo – výpis argumentů do konzole

reset – provedení resetu CPU

sleep – pozastavení běhu na určitou dobu

version – vypíše verzi monitoru

usb – (znovu)detekuje USB zařízení

bmp – zobrazení bmp obrázku do framebufferu

automount – automaticke připojení adresáře při prvním přístupu

readline – načtení řádku z konzole

timeout – počkání jestli nastane timeout (lze přerušit vstupem uživatele)

unlzo – Dekomprimuje komprimovaný LZO soubor

Příkazy pro práci s Flattened Device Tree

oftree – načtení, dupování a uvolnění DeviceTree blobů

of_node – vytváření a mazání uzlů v DeviceTree

of_property – vytváření, modifikace a mazání vlastností DeviceTree uzlu