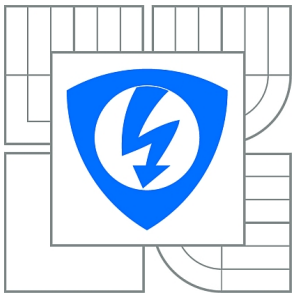
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

EMBEDDED SYSTÉM PRO SBĚR DAT

EMBEDDED SYSTEM FOR DATE COLLECTION

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

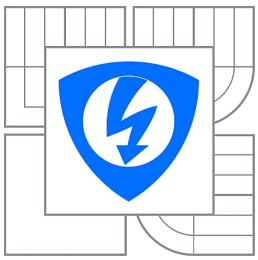
AUTOR PRÁCE — Bc. KAMIL VARGA

AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ MACHO, Ph.D.

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Kamil Varga	***ID:***	83163
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Embedded systém pro sběr dat

POKYNY PRO VYPRACOVÁNÍ:

1. Prostudujte problematiku návrhu a implementace embedded systémů založených na OS Linux určených pro sběr dat. Zabývejte se i problematikou souborových systémů.
2. Seznamte se s databázovými systémy a web servery určenými pro embedded aplikace.
3. Vyberte vhodnou distribuci OS Linux, databázový systém a web server. Navrhněte, implementujte, a odlaďte softwarové vybavení embedded systému , který by umožňoval sběr dat z AD převodníků a jejich ukládání do databáze umístěné v embedded systému. Komunikaci s embedded systémem realizujte prostřednictvím HTTP a SSH protokolu. HTTP protokol by měl sloužit zejména pro získávání uložených dat, SSH protokol pro administraci a konfiguraci systému.

DOPORUČENÁ LITERATURA:

[1] O'KEEFE, Greg. How To Build a Minimal Linux System from Source Code [online]. v0.8. September 2000[cit. 2010-01-25]. Dostupné z: http://axiom.anu.edu.au/~okeefe/p2b/buildMin/buildMin.html
[2] BEEKMANS, Gerrard. Linux From Scratch [online]. Version 6.5. 2009 [cit. 2010-01-25]. Dostupné z: http://www.linuxfromscratch.org/lfs/view/stable/.

[3] Šimůnek Milan. SQL kompletní kapesní průvodce. Praha: Grada,1999.247 s.ISBN80-7169-692-7.

Termín zadání:	8.2.2010	***Termín odevzdání:***	24.5.2010

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

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

UPOZORNĚNÍ:

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřící techniky

EMBEDDED SYSTÉM PRO SBĚR DAT

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

ABSTRAKT

Předkládaná diplomová práce řeší problematiku embedded systémů a jejich vývoje. Dále se zabývá operačním systémem GNU/Linux a jeho křížovou kompilací. Hardwarová problematika je také rozebrána. Konkrétním řešením je systém, který je schopen startovat z USB disku a je postaven na distribuci OpenWrt běžící na routeru Edimax BR-6104KP. Celé toto řešení je popsáno tak, aby z něj mohl být odvozen jakýkoliv obecný embedded systém.

KLÍČOVÁ SLOVA

Embedded systém, Sběr dat, GNU/Linux, OpenWrt, Křížová kompilace, Edimax BR-6104KP, HTTP Server, SSH Server, Embedded databáze, A/D převodník, SPI, GPIO, Python, SQLite

Brno 2010 **Bc. Varga Kamil**

BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering and Communication

Department of Control and Instrumentation

EMBEDDED SYSTEM FOR DATA COLLECTION

Supervisor: **Ing. Tomáš Macho, Ph.D.**

ABSTRACT

This master's thesis is describing problems of embedded systems and their development. Furthermore, the operating system GNU/Linux and it's cross-compiling is described. Hardware problems are analyzed too. Concrete solution is system which is able to boot from the USB disc and it is based on OpenWrt distribution running on Edimax BR-6104KP router. The whole solution is described the way which can lead to any general embedded system.

KEYWORDS

Embedded system, Data collection, GNU/Linux, OpenWrt, Cross-compiling, Edimax BR-6104KP, HTTP Server, SSH Server, Embedded Databases, A/D converter, SPI, GPIO, Python, SQLite

Brno 2010 **Bc. Varga Kamil**

Bibliografická citace

VARGA, K. *Embedded systém pro sběr dat*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 87 s., 7 příloh. Vedoucí práce Ing. Tomáš Macho, Ph.D.

Prohlášení

„Prohlašuji, že svou diplomovou práci na téma Embedded systém pro sběr dat jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

V Brně dne: **24. května 2010** ……
podpis autora

Poděkování

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

V Brně dne: **24. května 2010** ……
podpis autora

OBSAH

1. ÚVOD

Předkládaná práce si klade za cíl seznámit čtenáře komplexně s problematikou návrhu a tvorby embedded systémů se zaměřením na operační systém GNU/Linux. Celá práce je rozdělena do čtyřech úrovní v nichž je čtenář stále hlouběji seznamován s vývojem nového systému.

Na první úrovni, jež se věnuje kapitola dvě, je uveden obecný úvod do embedded systémů a jejich problematiky a to jak po hardwarové, tak softwarové stránce. Na tento úvod volně navazuje další kapitola jež se zabývá systémovým návrhem již konkrétního embedded systému. V této kapitole je uveden koncept, jakým by se měl ubírat embedded systém pro sběr dat. V kapitole následující jsou již uvedeny konkrétní volby hardwarových změn, zvoleného softwaru a také operačního systému. Ke každé jednotlivé volbě je uveden popis, objasňující co k jejímu konkrétnímu nasazení vedlo. Hlubší popis je potom věnován operačnímu systému GNU/Linux. Popis těchto důležitých součástí se soustředí na základy nutné k pochopení dalších konkrétních voleb a také práci systému. V nadcházející kapitole a poslední úrovni je již konkrétní postup zprovoznění hardwarové a poté i softwarové stránky zcela nového embedded systému určeného ke sběru dat.

V každé kapitole jsou mimo konkrétního výběru nastíněny i další možnosti, jakými se může samotný vývoj ubírat. V práci tak není nucen pouze jeden směr, ale je uvedena jedna konkrétní implementace a k ní další řada příkladů možných cest.

2. VŠEOBECNÁ PROBLEMATIKA EMBEDDED SYSTÉMŮ

V samotném začátku je vhodné definovat co to vlastně embedded systémy jsou. Za slovíčkem „embedded“ pocházejícím z angličtiny a ve volném překladu znamenajícím „ukotvený, usazený“ se skrývají zařízení, která se hlavně s rozmachem mikroelektroniky staly běžnou součástí našich životů. Jsou jimi například mobilní telefony, smartphony – chytré telefony, PDA, ale také routery, systémy ABS v automobilech, herní konzole, FBW systémy v letadlech a spousta dalších zařízení. Jedná se tedy o zařízení, jež mají svůj vlastní procesor, paměť a nějaké periferie. Tyto jsou programem čteny a ovládány tak, aby plnily účel ke kterému jsou určeny. Jedná se vlastně o počítačový systém, který ovšem není klasickým počítačem, ale koncovým zařízením pro jeden konkrétní účel. Z této „definice“ lze potom lépe i pochopit anglický výraz embedded.

Po krátkém seznámení s embedded systémy je nyní možné zabývat se tím co je třeba při vývoji embedded systému uvažovat a na co se soustředit. I když je každé embedded zařízení jiné a v jistém smyslu originální, tak se i přes to vývoj každého takového zařízení neobejde bez určitých základních kroků a to jak v otázkách volby hardware, tak i volby software. Tyto jsou mezi sebou úzce spjaty, protože jakmile není například dostatek systémových prostředků pro běh daného software – typicky paměť, čas jednotky CPU, tak daný software prostě fungovat nemůže. Stejně i naopak když je systémových prostředků dostatek, ale žádný software, který by je uměl využít nebo ovládat, tak to také nevede k úspěchu. Proto vývoj hardware a software pro embedded zařízení jde většinou „ruku v ruce“. Nicméně i tak jsou vždy v počátku nastaveny určité meze a to většinou převážné po ekonomické stránce, ale v moderní době také čím dál více populární omezení velikostní – každý výrobce se snaží svůj produkt snaží udělat co nejmenší a s co nejmenší spotřebou. Tyto meze následně většinou značně ovlivní výběr a návrh hardware a do něj se musí poté „vměstnat“ i software. Ač jsou tato omezení nepopulární hlavně pro vývojáře, tak mají i své výhody. A to ať už třeba z pohledu ekologického – není zbytečně plýtváno vzácnými kovy a dalšími zdroji při výrobě a nebo i z pohledu posunu vývoje – je zde

větší prostor pro myšlení a vznikají tak nové nápady a inovace - jak se dá ušetřit místo v paměti, čas potřebný k výpočtu na CPU a další. Bližšímu rozebrání možných hardwarových voleb a otázek v návrhu embedded systému se již věnuje následující kapitola.

2.1 HARDWARE PRO EMBEDDED SYSTÉMY

Hardwarová stránka každého embedded zařízení je to, co má největší vypovídající schopnost o celkovém výkonu a možnostech daného embedded zařízení. Přičemž se jedná hlavně o tzv. „systémové prostředky“, tedy paměť, výkon CPU a další. Na samotném počátku vývoje hardware by si měl každý vývojář nebo vývojový tým položit následující otázky:

- Co všechno na embedded systému poběží – jaké to bude vyžadovat systémové prostředky?
- Bude embedded zařízení modulární – bude dovolovat další rozšíření?
- Existuje již nějaké podobné zařízení?
- Jaká platforma CPU bude nejoptimálnější – příkon, možnosti, podpora OS?
- Jaká rozhraní budou zapotřebí?
- Jaká má být koncová cena?

Odpovědi na tyto otázky mohou vývojovému pracovníkovi nebo vývojovému týmu značně ulehčit práci při samotném vývoji a tvorbě hardwarové části. Hned odpověď na první otázku je asi tou nejdůležitější a vychází z tzv. systémového návrhu. Systémový návrh je základní design a představa toho, jak by měl koncový embedded systém vypadat a pracovat. Je velice důležité, zvláště u větších projektů což většinou embedded systémy jsou, hned z počátků definovat koncept a hrubý návrh celého systému. Z tohoto konceptu se bude vycházet jak při návrhu hardware, tak při návrhu software. Jakmile je proveden návrh embedded systému a je tedy jasno k čemu bude sloužit, resp. co na něm poběží, tak lze systémové prostředky „optimalizovat“ na míru těmto vlastnostem s tím, že stačí, aby byla zaručená dostatečná rychlost odezvy a práce. Volí se tedy systém, který bude mít dostatečně rychlé CPU a dostatečně velikou paměť RAM a případně i pevnou paměť (flash, HDD, ROM) tak, aby na něm software běžel spolehlivě a měl i drobnou rezervu.

Odpovědí na tuto otázku se tedy mohou značně snížit náklady jak na vývoj – počáteční návrh by již neměl vyžadovat velké úpravy a neměly by nastat kompromisy v řešení což by mělo zaručit i vyšší spolehlivost zařízení, tak i na samotnou výrobu – zařízení využívá téměř zcela všechny své prostředky a náklady na výrobu jsou tedy prakticky na minimu. Odpověď na druhou otázku má úzkou souvislost s první. Jestliže by zařízení mělo být i modulární, tedy zařízení, které lze v budoucnu i rozšířit, tak je třeba mu přidělit více systémových prostředků za cenu toho, že některé z nich nebudou třeba nikdy využity. V tomto případě je třeba volit množství systémových prostředků s ohledem na odhad budoucích rozšíření s přidanou další rezervou. S odpovědí na další otázku přichází mnohdy i značná úspora času při vývoji. Jestliže totiž už existuje podobné zařízení i když třeba určené pro jiný účel (to bude i případ tohoto textu dále) a není chráněno patentem či autorskými právy (tyto dvě podmínky lze opominout jedná-li se o zařízení pro nekomerční účely), tak jej lze jednoduše zkopírovat a upravit dle kýžených potřeb. Odpovědi na tři poslední z před-vývojových otázek už jsou spíše blíže určující pro konkrétní cílový koncept, což se hodí zvláště v případě více možných návrhů a cest. Koncová cena zařízení je posledním faktorem, který může zvláště v případě návrhu na „míru“ ještě pohnout s konečnými parametry embedded systému a snížením rezerv a většinou se tak také děje na úkor software. Jeho vývoj je poté sice pracnější (je nutné kód více optimalizovat), nicméně na rozdíl od hardware se jeho cena promítne do výsledku pouze jednou a rovnoměrně rozloží mezi všechny takto vyprodukované zařízení. Toto je obrovská výhoda zvláště při masové výrobě.

Mozkem každého embedded systému je jádro nebo i více jader CPU tedy centrální procesorové jednotky a je proto zapotřebí mu věnovat zvýšenou pozornost. CPU je dnes poměrně široká škála. Jak je uvedeno v [15], tak některé z nich se hodí pouze pro některé aplikace, jako třeba signálové procesory – DSP nebo mikrokontroléry, jiné se mohou použít prakticky kdekoliv a těmi jsou procesory pro všeobecné použití. Těmito procesory započala celá éra dnešních procesorů a jsou od nich odvozeny i další výše zmíněné typy procesorů. Mikrokontroléry vznikly z potřeby integrovat již některé z periferií jako součást CPU. DSP procesory poté pro zpracování některých speciálních matematických operací v reálném čase, např.

rychlou Furierovu transformaci – FFT. V embedded zařízeních se mohou vyskytovat všechny typy uvedených procesorů. Nejedná-li se o speciální aplikace, které navíc většinou nevyužívají ani podporu operačního systému, tak jsou využívány převážně procesory pro všeobecné použití. Co se týká podpory operačního systému – OS, tak ta je u procesorů víceméně dána podporou vyšších programovacích jazyků – C, C++ a další, dostatečně širokou instrukční sadou a existencí jednotky pro správu paměti. I když například distribuce GNU/Linuxu zvaná µLinux si poradí i s mikrokontroléry bez této jednotky. Některé mikroprocesory používané v kritických odvětvích, jako je například letectví navíc musí být i certifikované. V dnešní době nejpoužívanější architektury procesorů pro všeobecné použití jsou následující:

- MIPS
- x86, x86-64
- ARM
- PowerPC
- Alpha
- Sparc

Za zmínku stojí hlavně architektura ARM a MIPS, které se v embedded zařízeních využívají asi nejčastěji. Na poli stolních PC se jedná o nejrozšířenější architekturu v případě x86 a v dnešní době stále populárnější x86-64. Pro bližší studium uvedených architektur lze použít jako zdroj například wikipedii, popřípadě dokumentaci výrobce ke konkrétnímu typu mikroprocesoru.

Jak již bylo uvedeno výše, tak embedded systémy lze vyvíjet i z již nějakého hotového zařízení. Nejvhodnějšími adepty pro tento vývoj jsou následující zařízení:

- vývojové desky
- routerboardy a jiná zařízení s „otevřeným“ kódem
- jiné zařízení s již běžícím operačním systémem.

V případě vývojových desek se jedná o návrhy většinou přímo od výrobce procesoru, ale lze se také setkat s vývojovými kity pro výrobu embedded zařízení. Tato zařízení mají tu výhodu, že jsou přímo určená pro nasazení v embedded systémech a jsou proto připravena i na implementaci operačního systému a také jsou plně dokumentována. Jejich nevýhodou potom může být cena, která je poměrně vysoká a

proto nejsou příliš vhodné pro vývoj jednoho koncového zařízení, ale spíše pro vývoj platformy, která bude vyráběna masově.

Routerboardy a další zařízení s „otevřeným“ kódem tzv. „open source“ jsou taktéž velice vhodná. Jedná se o zařízení, jež jsou velice podobná vývojovým deskám, pouze s tím rozdílem, že je na nich již nasazen operační systém a nějaký software. Jejich možnosti rozšíření a nasazení jsou menší než v případě vývojových desek. Jejich výhodami jsou potom opět dokumentace tentokráte ovšem i již k implementaci operačního systému, také variabilita, i když menší než v případě vývojových desek a nakonec cena, která bývá o poznání nižší. Tato cesta je velice vhodná k vývoji jednoho konkrétního zařízení.

Posledním typem jsou zařízení, u kterých výrobce pouze uvede, že obsahují operační systém, ale již neuvede o jaký systém se jedná, či jakým způsobem je implementován. V tomto případě se vývojová cesta stává nejtěžší a stojí za zvážení zda-li se více nevyplatí kompletní vývoj. To ovšem platí, pokud se žádná z předchozích dvou cest nedá použít. V takovémto případě totiž přichází na řadu měřící technika a hluboká analýza celého zařízení. Tento postup lze většinou vidět u nadšených techniků a kutilů, kteří si pořídí nějaké zařízení, ale rádi by poupravili jejich vlastnosti. Pro cestu vývoje od úplného začátku tato cesta není tou nejlepší volbou a spíše ji nelze doporučit, pokud člověk opravdu neví co dělá.

Dalším důležitým faktorem při vývoji HW embedded systému je volba rozhraní, jež budou propojovat „jádro“ systému s periferiemi. V případě vývoje od úplného začátku je třeba rozhraní nejprve zvolit a poté správně zaimplementovat do návrhu hardware. V případě již použitého zařízení vybrat takové, jež potřebné rozhraní již implementuje nebo ho lze rozšířit takovým způsobem, že rozhraní lze použít. Volba rozhraní je daná hlavně volbou periferií, jež jsou plánovány pro připojení k embedded systému. Tyto periferie také mohou značně ovlivňovat potřebné systémové prostředky. Například barevný dotykový display s větším rozlišením(např. 800x600) a plnou podporou barev – 24 bitů jistě nebude možné patřičně využít na embedded systému s 16 MB pamětí RAM. Samotný buffer pro vykreslování obrazu na takovémto zařízení je zapotřebí o velikosti 800x600x24 = 11 520 000b = téměř 11MB dat. Z tohoto příkladu je jasně vidět, že návrh hardware pro

embedded systém je komplexní záležitostí a vyžaduje vhodnou volbu hlavně rychlosti procesoru, velikosti paměti RAM, pevné paměti (např. FLASH, ROM, nebo lze použít i jiné sítí připojené zařízení), použitých periférií a rozhraní. Dnešními asi nejčastějšími rozhraními používanými v embedded systémech jsou:

- GPIO
- SPI
- RS-232, IEEE 1284
- CAN
- Ethernet
- USB
- PCI, PCI mini, PCI-express
- ATA, S-ATA
- D-SUB, DVI, HDMI, Composite Out

GPIO rozhraní je rozhraním samotného procesoru, jedná se o vstupně – výstupní porty pro všeobecné použití a tyto se také používají pro implementaci dalších systémových sběrnic či rozhraní.

SPI sběrnice je sériová linka přímo implementována v CPU a jedná se pouze o jiné chování vstupně – výstupních portů s přímou podporou v jádře – stačí je pouze patřičnými registry nastavit a aktivovat.

RS-232 a IEEE 1284 jsou rozhraními pro sériový resp. paralelní přenos dat. V dnešní době již nejsou tolik využívána, protože bývají nahrazeny rychlejšími rozhraními. V dřívější době se používali pro komunikaci s měřícími přístroji, snímači a dalším.

Sběrnice CAN je jedním z průmyslových standardů nasazených například v letectví a automobilech pro komunikaci mezi hlavním „mozkem“ embedded systému a menšími jednotkami, jež jsou připojeny přes tuto sběrnici (například spojení jednotek pro ovládání klapek křídel a centrální řídící jednotkou).

Standard ethernet je spojen se síťovou komunikací. Slouží nejen ke komunikaci po celosvětové síti internet, ale také pro spojení podobné popisovanému u sběrnice CAN.

Sériové rozhraní USB se dá použít a je využíváno pro odpojitelné periferie všeho druhu od síťových karet, přes pevné disky až po webové kamery a měřící přístroje a další. Jeho velkou výhodou jsou vysoké rychlosti, kterých dosahuje a také poměrně velké vzdálenosti mezi embedded systémem a připojenými periferiemi. Vývoj zařízení na toto rozhraní sice již není tak triviální záležitostí, jako bylo například v případě pro sběrnici RS-232, ale klady tohoto rozhraní nad touto nevýhodou jednoznačně zvítězili.

Sběrnice PCI, PCI-mini a PCI express jsou stejně jako USB známé převážně ze stolních počítačů, nicméně jsou dnes používané i v embedded zařízeních (hlavně PCI-mini). Jejich výhodou jsou velké přenosové rychlosti a slouží téměř výhradně ke zpracování většího množství dat, například obrazových, síťových a dalších. Ačkoliv sběrnice PCI express nemá se sběrnicemi PCI prakticky nic společného mimo názvu, tak je to právě ona, která vytlačuje PCI díky obrovským přenosovým rychlostem a sériovému přenosu dat.

Rozhraní ATA a S-ATA se používají pro připojení disků a optických mechanik. ATA rozhraní je staršího data a jedná se o paralelní rozhraní, v dnešní době bývá vytlačováno novějším S-ATA rozhraním, které je sériové a v poslední verzi (3.0) dovoluje přenosy až 600MB/s, což je oproti ATA, která má maximální rychlosti okolo 133MB/s velice slušný nárůst.

Poslední uvedené typy rozhraní jsou rozhraní pro přenos obrazových (v případě HDMI i zvukových) dat do zobrazovačů jako jsou monitory, projektory, panely a další. D-SUB a Composite Out jsou analogová rozhraní a tudíž přenášejí čistý analogový signál, na proti tomu stojící modernější DVI a v poslední době stále populárnější HDMI jsou rozhraními digitálními.

Výše uvedený přehled popisuje pouze základní vlastnosti rozhraní. Přesnější popis by byl nad rámec této práce. Případný zájemce o nasazení některého z daných rozhraní by měl nastudovat buďto přímo normu nebo literaturu zabývající se touto problematikou.

Tímto je završen obecný úvod k problematice návrhu hardware pro použití v embedded systémech. Pro konkrétní možnosti nasazení uvedených pouček

je zde kapitola zabývající se návrhem hardware pro embedded systém určený ke sběru dat, kde je rozebráno vícero návrhů tohoto systému.

2.2 SOFTWARE PRO EMBEDDED SYSTÉMY

Podobně jako v případě hardware, tak i u software se lze vydat cestou úplně od nuly nebo použitím již existujících nástrojů. V případě vývoje úplně od nuly půjde o naprogramování celého embedded systému bez využití operačního systému. Tato cesta je šitá plně na míru daného hardware a pokud je zařízení správně naprogramováno, tak plně využívá jemu přidělené systémové prostředky. Výhodou je tedy nejlepší možné využití systémových prostředků – neběží nic, co by nebylo potřeba a také velikost programu je přesně taková, jaká je nutná ke správné funkci zařízení. Nevýhodou této koncepce je potom dlouhý vývoj a prakticky žádná modularita – jakmile je třeba změnit nějakou vlastnost nebo v horším případě část hardware, tak je třeba změnit větší část kódu a celý systém překompilovat a znovu nahrát do zařízení. Dnes se tento způsob vývoje využívá hlavně u zařízení se speciální funkcí nebo u jednodušších zařízení, kde vývoj takového software není tak pracný. Oproti této cestě stojí možnost využití operačního systému. Tedy software, který se stará o využití a správu systémových prostředků a programátorovi přináší abstraktní prostředí. Programátor se potom nemusí starat o konkrétní použitý hardware, ale pouze o abstrakce, jež mu operační systém poskytuje. Vzniká tak software, který může být přenositelný na více platforem. Vykoupením za tyto klady je potom větší náročnost na systémové prostředky, protože jádro operačního systému musí být napsáno tak, aby běželo na jakékoliv platformě buďto úplně beze změn nebo s minimálními změnami. S jakými operačními systémy se dnes tedy lze nejčastěji setkat?

- Microsoft Windows CE
- Microsoft Windows XP Embedded
- Symbian
- Google Android
- Solaris
- BSD
- GNU/Linux

Systémy Microsoft Windows jsou poměrně dosti náročnými systémy, které se vyznačují okenním manažerem, pomocí kterého se pracuje prakticky s celým systémem. Hardware na kterém takovýto systém poběží musí tedy obsahovat zobrazovací jednotku i některou z periferii pro ovládání. Tyto systémy jsou nejčastěji nasazovány na různé smartphony a touchpady. Jejich velkou nevýhodou je, že se jedná o komerční software s uzavřeným kódem, což znemožňuje jakékoliv zásahy do jádra systému. Jako alternativy reprezentující open source software jsou uvedeny další operační systémy.

Symbian operační systém stejně jako systémy softwarové společnosti Microsoft je využíván převážně na mobilní platformách a to převážně na architektuře ARM. Do ne dávna byl také komerčním operačním systémem s uzavřeným kódem. Dnes je již jeho kód zveřejněn pod jednou z open source licencí. Tento systém je systémem reálného času, což znamená, že dokáže zpracovávat operace tak, že se jeví jakoby běžely v reálném čase. V systémech, které neumí pracovat v reálném čase není zaručená odezva jádra a může se tak stát, že stejná část programu bude pracovat v různé chvíle různě dlouhou dobu.

Jako poslední ze systémů víceméně orientovaných na mobilní platformy je operační systém společnosti Google zvaný Android. Tento systém je postaven na GNU/Linuxu a od první chvíle jeho uvedení (pod open source licencí) je o něj velký zájem (ostatně jako o jakýkoliv jiný produkt společnosti Google) a jeho nasazení ve svých přístrojích zkouší stále více společností. Na poli osobních počítačů je v tuto chvíli vyvíjen společností Google také další operační systém - Google Chromium, taktéž založen na GNU/Linuxu slibující velké věci a změny.

Systém Solaris od firmy Sun existuje ve verzi komerční, ale i open source zvané OpenSolaris. Jeho primární platformou jsou procesory architektury Sparc, které taktéž vyvinula firma Sun, ale je portován i na nejrozšířenější architekturu stolních počítačů a to x86 a x86-64. Solaris vychází stejně tak, jako i další dva dále zmíněné systémy z rodiny Unixových systémů, v angličtině zvaných „Unix-like“.

BSD systémy vycházejí taktéž z Unixu jakožto operačního systému a jsou nejvíce využívány asi v serverových aplikacích. Na poli embedded zařízeních jsou taktéž hojně využívána díky jejich obsáhlé dokumentaci a portu na velké množství

architektur. Jejich nasazení je velice podobné GNU/Linuxu a systém jako takový také dovoluje spouštět programy určené původně pro GNU/Linux.

GNU/Linux systém vznikl jako open source projekt založený na Unixovém systému. I když na poli osobních počítačů mu patří jen zlomek trhu, který je z největší části obsazen operačními systémy firmy Microsoft, tak v embedded zařízeních a v superpočítačích se GNU/Linux využívá jako většinový. Jeho nespornou výhodou je stabilní, rychlé jádro, které je navíc portováno prakticky na všechny známé architektury a to včetně systémů bez jednotky pro správu paměti (MMU) – výše zmiňovaný µLinux. Existuje také verze jádra pro real time systémy. GNU/Linux systém bude také použit pro vývoj embedded systému pro sběr dat uvedeném dále.

Jelikož embedded systémy vládnou omezenými systémovými prostředky, tak i software instalovaný na embedded systémy společně s jádrem operačního systému bývá značně rozdílný od svých „bratrů“ z PC. Charakteristickými vlastnostmi software pro embedded systémy jsou jednoduchost, malá velikost, nenáročnost na procesorový čas a malá spotřeba operační paměti – v angličtině: „memory footprint“. Proto na embedded systémech většinou nenajdeme klasické PC programy ve stejné podobě, ale většinou se jedná o speciální upravenou verzi téhož programu ochuzeného o některé vlastnosti. Některé z těchto vlastností lze později při-instalovat ve formě modulů. Zdárným příkladem takovéhoto počínání může být například interpret Python-mini. Na PC interpret Python zabírá několik desítek MB, kdežto v případě Python-mini se jedná o necelých deset MB. Rozdíl je v tom, že Python-mini neobsahuje moduly, které nejsou ve větší množině případů používány. Taktéž je odebrána dokumentace a celý interpret se omezuje jen na to nejzákladnější. V tomto případě se počítá s tím, že bude interpret Pythonu sloužit pouze pro jednoduché administrátorské operace – tvorba skriptů, nikoliv komplexních programů. Toto byl případ programu, kde standardní program z PC byl portován v upravené podobě i na embedded. V dalším případě se jedná o množinu programů se kterými se na PC většinou nesetkáme. Jsou to programy, jež plní stejnou funkci, jako programy známé z PC, pouze s jistými omezeními a podstatně menším využitím systémových

prostředků. Příkladem budiž program dropbear. Jedná se o SSH server, který slouží ke vzdálené správě a který je díky svým nárokům používán na embedded systémech.

Z výše uvedeného je patrné, jak je volba software velice důležitou součástí vývoje embedded zařízení a je jí třeba věnovat zvýšenou pozornost . Je třeba vybrat jen to, co je skutečně třeba, taktéž jádro je nutné upravit (nakonfigurovat) jen pro funkce, jež budou opravdu potřeba, aby se zamezilo situaci, kdy dojdou přidělené systémové prostředky a celý systém se „zhroutí“ nebo dojde ke snížení odezvy na nepřijatelnou úroveň. Výběr software úzce souvisí s volbou hardware a to hlavně se zvolenou architekturou, velikostí operační paměti a pevné paměti, protože ne všechny programy jsou portovány na všechny architektury a se zanedbatelnou spotřebou paměti.

3. KONCEPT EMBEDDED SYSTÉMU PRO SBĚR DAT S WEBOVÝM ROZHRANÍM

Celá následující kapitola se zabývá již konkrétnějším návrhem embedded systému a to konkrétně systému, který bude schopen sbírat předem neurčená data, například z A/D převodníků, bude je ukládat do databáze, kterou bude možné zobrazit pomocí webového rozhraní celosvětové sítě Internet. Navíc bude tento systém možno vzdáleně konfigurovat a kontrolovat pomocí protokolu SSH.

Na samotném začátku je nutné definovat tzv. systémový návrh embedded systému pro sběr dat. O tento návrh bude postaráno v následujících dvou kapitolách. Tučně budou v textu zvýrazněny systémové požadavky a zbytek textu bude sloužit pro vysvětlení konkrétních požadavků. V první podkapitole budou rozebrány systémové požadavky pro softwarovou část, ve druhé potom pro hardwarovou.

3.1 SYSTÉMOVÝ NÁVRH SOFTWARE EMBEDDED SYSTÉMU

Již z představy toho, co má embedded systém dělat je patrné, že **bude implementován operační systém**. Ne, že by nebylo možné implementovat webový server, databázi a SSH server zcela novým programem běžícím na daném zařízení, ale budeme v tomto případě šetřit vývojáře software, kteří by se určitě opravdu „zapotili“ (jednomu vývojáři by tato práce zabrala dokonce určitě několik týdnů, či spíše měsíců) při implementaci všech těchto vlastností a zvolíme implementaci OS. Již touto volbou je zredukováno množství vývojových pracovníku SW části na minimum a taktéž je poskytnuto abstraktní rozhraní OS, které zaručuje minimální úpravy kódu v případě, že bude zapotřebí upravit hardwarové vlastnosti. Také je zcela zřejmé, že **nebude zapotřebí žádné zobrazovací zařízení**, protože bude vše možné kontrolovat vzdáleně pomocí SSH a zobrazení bude probíhat použitím HTTP serveru. Naopak **bude zapotřebí** nějaké **síťové rozhraní a** také **rozhraní pro připojení zařízení jež budou poskytovat sbíraná data**. Dále **bude zapotřebí** nějaké **větší paměťové zařízení**, aby mohla být šířka uložených výsledků v databázi dostatečná.

Nyní k softwarové stránce věci. Je zřejmé, že spolu **budou** muset nějakým způsobem komunikovat **databáze**, hardwarové rozhraní a **HTTP server**. Tato komunikace může být provedena pouze nějakým programem. Program může být napsán ve vyšším programovacím jazyce – C nebo C++ nebo v nějakém z tzv. skriptovacích či interpretovaných jazyků. Interpretované jazyky nabývají postupně na popularitě a používají se ve stále větší míře. Jejich zařazení mezi interpretované vyplývá z potřeby běžícího interpretu, který příkazy ve skriptu či programu zpracovává a provádí. Výhodou těchto jazyků je zvýšená efektivita programování, protože o většinu systémových záležitostí, jako je dynamická alokace paměti, určení typu proměnné a další, se již stará interpret a program tvořený v některém z těchto programovacích jazyků je podstatně jednodušší než například v jazyce C či C++. Nevýhodou potom je mírné snížení rychlosti na rozdíl od třeba již zmíněného C či C++. Pro další snížení softwarové náročnosti výsledného embedded systému **bude** tedy **implementován interpretovaný jazyk**. Konfigurace celého systému **bude** možná přes **SSH server a** také přes SPI rozhraní CPU, aby bylo možné uvést zařízení do chodu v případě nedostupnosti sítě. Tento požadavek se tedy týká také hardwarové části, kde musí být implementováno alespoň SPI rozhraní. Aby byla možná spolupráce HTTP serveru a interpretovaného jazyku, tak musí být použit **HTTP server s podporou protokolu CGI**.

Tímto lze systémový návrh SW části označit za uzavřený a lze přejít k systémovým požadavkům pro volbu hardware.

3.2 SYSTÉMOVÝ NÁVRH HARDWARE EMBEDDED SYSTÉMU

Tato podkapitola se pokusí odpovědět na všechny otázky, jež byly formulovány v první kapitole a kterými je třeba se zabývat při vývoji hardware embedded systému. Nuže první otázkou je otázka běžícího software na embedded systému. I když tato otázka bude konkrétně zodpovězena až v další podkapitole, tak již ze systémového návrhu software lze odhadnout, jaké systémové prostředky budou zapotřebí. Jestliže není k dispozici dostatečné množství znalostí, co bude jádro OS, databáze a uvedené servery vyžadovat, tak se lze inspirovat u již hotových zařízení. Po průzkumu, resp. testu hotových zařízení vyplynulo, že pro běh jádra, HTTP

serveru a SSH serveru stačí necelých 12MB operační paměti. Nyní je také zapotřebí ještě paměť pro interpret a databázi. Tomuto by měly stačit 4MB paměti. Výsledně tedy **bude** stačit **16MB operační paměti** i když větší paměť nebude na škodu z důvodu rezervy. Architektura použitého CPU by měla být taková, aby na ni bylo jednoduché nasadit operační systém. Rychlost CPU by měla být také odvozena od systémových požadavků na software. V tomto konkrétním případě bude nejnáročnější zřejmě sběr dat a běh HTTP serveru. Jelikož nebylo určeno o jak rychlá data se má jednat bude zde předpoklad rychlostně nekritických dat a mělo by **stačit 100MHz CPU**. Architekturu bude vhodné zvolit některou z úsporných, efektivních a dostatečně podporovaných OS, například **ARM** či **MIPS** za cenu toho, že **implementovat** na ně operační systém bude mírně náročnější než třeba v případě x86 či x86-64. Výhodou naopak budou nízké náklady na provoz díky snížené spotřebě. Při řešení otázky rozhraní je zapotřebí uvažovat již výše zmíněné **SPI rozhraní**, dále bude zapotřebí síťové rozhraní, nejlépe ethernet. Pro sběr dat by bylo vhodné mít jak sériové, tak paralelní rozhraní. Toto může být realizováno vstupně – výstupní porty pro všeobecné použití – **GPIO**, které musí být pouze nějakým způsobem chráněny proti nesprávnému napětí a proudu na vstupech. Komunikace, ať již sériová či paralelní, může být realizována softwarově. Tato cesta bude jednodušší než implementovat navíc ještě další specializované obvody. Nicméně nic nebrání tomu použít rozhraní RS-232 či IEEE 1284. Co se týká paměťových nároků OS, tak ty se v případě pevné paměti pohybují pod 2MB, nicméně je nutné implementovat i databázi pro sběr dat a interpret, tak by bylo vhodné mít **alespoň 16MB pevné paměti**. Tuto lze připojit přímo k procesoru ve formě flash či dnes již méně obvyklé eeprom paměti nebo pomocí dalšího rozhraní USB. Rozhraní **USB je** tedy **volitelnou součástí**. **Koncová cena** takového zařízení by měla být samozřejmě co nejnižší a předběžným odhadem dle navržených hardwarových požadavků by se měla pohybovat **do 1500 Kč**. Jakmile jsou definovány hrubé požadavky na hardware, tak je vhodná chvíle na průzkum trhu, zda-li již podobné zařízení neexistuje. Po drobném internetovém průzkumu bylo odhaleno více vhodných zařízení, které splňují uvedené požadavky a lze se tedy vydat jednodušší cestou úpravy již existujícího zařízení. Následuje tabulka základních parametrů vhodných zařízení:

Deska:	Edimax BR6104KP	Beagleborad	BifferBoard	RouterStation
CPU:	ADM5120 MIPS/ 175MHz	ARM Cortex-A8 / 600MHz	Intel 486SX / 150MHz	Atheros MIPS 24k /680MHz
FLASH:	2MB	256MB	8MB	16MB
RAM:	16MB	128MB	32MB	64MB
Rozhraní:	2x SPI, 19x GPIO,1x JTAG,2x USB, 5x Ethernet	I2C, I2S, SPI, 2x MMC/SD, DVI-D, JTAG, S-Video, Stereo In/Out, USB, RS-232	USB, Ethernet, SPI, GPIO, JTAG	3x Ethernet, 3x mini-PCI, USB, SPI, GPIO, JTAG
Spotřeba:	max. 6W	2,5W	1W	max.7W
Orientační cena:	700,-	3500,-	1000,-	1000,-

Tabulka 1 Embedded systémy vhodné pro sběr dat

Prakticky kterýkoliv z těchto produktů by bylo možno použít, nicméně z výše uvedených systémových požadavků se jeví jako nejvhodnější volba routerboard firmy Ubiquity nebo router firmy Edimax. Beagle Board by byla skvělou volbou pro zařízení, které by muselo zpracovávat data v reálném čase a o velkém datovém toku. Například zvuková či obrazová data. Jelikož ovšem povaha dat není definována, tak jsou předpokládána data, která nebudou náročná. Tento předpoklad byl již výše zmíněn, nicméně jej lze kdykoliv změnit a vydat se cestou právě uvedené desky Beagle Board, která je opravdu bohatá jak na výkon, tak na množství rozhraní. Spotřeba k tak obrovskému výkonu jakým deska disponuje je impozantní a cenová relace je taktéž přijatelná. Vzhledem k řečenému je nutné se tedy rozhodnout mezi routerem a routerboardem. V případě **router**u bude zapotřebí zprovoznit navíc i USB rozhraní, aby bylo možné připojit rozšiřující paměť, jelikož současná velikost paměti flash – 2MB je silně nedostačující. Jelikož toto zařízení bylo pořízeno ve chvíli, kdy varianta s routerboardem nebyla známa, tak **bude použit**o **jako embedded systém pro sběr dat**. Pro šířku záběru této práce bude toto řešení prospěšnější z toho důvodu, že bude uvedeno více implementačních detailů a proto bude možno

poznatky z této práce aplikovat na větší množství embedded zařízení. Následující tabulka shrnuje všechny uvedené systémové požadavky přehlednější formou:

Hardwarové požadavky:
Minimálně 16MB RAM
Minimálně 100MHz CPU - MIPS nebo ARM
Žádné zobrazovací zařízení
Větší disk – alespoň 16MB
SPI rozhraní
GPIO rozhraní
Ethernetové rozhraní
Volitelně USB
Router Edimax
Softwarové požadavky
Operační systém
Interpretovaný jazyk
HTTP Server s podporou CGI
SSH Server
Databáze

Tabulka 2 Systémové požadavky

Jakmile jsou nadefinovány i systémové požadavky na HW, tak lze přejít ke konkrétnímu výběru SW i HW částí, která jsou uvedeny v následující kapitole.

4. VOLBA HARDWARE A SOFTWARE

Celá čtvrtá kapitola popisuje proces voleb a možností vedoucí ke konkrétní aplikaci a cílovému systému. Kapitola je rovněž, jako i kapitoly předchozí, rozdělena do dvou hlavních částí. První z nich se zabývá hardwarovými rozšířeními a úpravami zvoleného zařízení, druhá část potom volbou software nasazeného na koncovém zařízení. Všechny volby jsou blíže rozebrány a také obohaceny o povědomí o dalších možnostech, které by mohly být použity.

4.1 HARDWARE EMBEDDED SYSTÉMU PRO SBĚR DAT

Kapitola zabývající se jednotlivými volbami hardware v sobě zahrnuje popis výběru a volby hardwarových rozšíření jež jsou zapotřebí pro danou aplikaci. Tyto změny obsahují řešení komunikace mezi stolním PC a cílovým zařízením, dále oddělením periférií a desky routeru a nakonec volbou obvodu pro sběr dat.

4.1.1 Komunikace PC/Router

Aby bylo možno do desky routeru nahrát nový software vytvořený dále, tak je prvně nutné spojit PC na kterém bude nový software vytvořen s routerem pomocí některého se standardních rozhraní. Jak je uvedeno například na stránkách [17], tak po rozebrání se na desce lze setkat s rozhraním JTAG a SPI (viz. Obrázek 1) určenými k tzv. „naflashování“ obrazu systému do paměti.

Obrázek 1 Rozhraní JTAG a SPI

Jinými slovy lze přes tato rozhraní změnit již fungující software a nahrát nový. Proto je zapotřebí jedno z těchto rozhraní nějakým způsobem spojit s PC, na kterém je nový software pro embedded systém vytvářen. Rozhraní JTAG lze připojit pomocí paralelního portu tiskárny. SPI potom pomocí rozhraní USB nebo sériového rozhraní RS-232. Pro všechna tato rozhraní je nutné použít převodník, který se stará o změnu napěťových úrovní nebo zpracování signálu na uvedených portech PC. Nejjednodušším a nejlevnějším z nich je převodník pro RS-232 port. Tento převodník se stará pouze o změnu napěťových úrovní, protože rozhraní RS-232 funguje na napětích až +/- 15V, které by paměť flash zničili. Jelikož byl zmíněný převodník také úspěšně použit, jak je například uvedeno na [16], tak bude použit i pro tento projekt.

4.1.2 Komunikace Router – periférní zařízení

Sběr dat je možno realizovat více možnými zařízeními na různých rozhraních. Samotný router disponuje rozhraními SPI, JTAG, USB a nepřímo vyvedenými porty GPIO. Jeden SPI port je jak již bylo zmíněno výše vyveden přímo na jednom z interních konektorů desky a slouží pro komunikaci a k nahrávání nového software. I když jej lze využít přímo v běžícím systému jako rozhraní, které lze softwarově ovládat, tak bude ponechán pro komunikaci s operačním systémem pro případ výpadku sítě nebo jiných potíží. Bude také využit při prvotním nastavení pro zprovoznění SSH komunikace. Druhý port není přímo vyveden, ale lze jej vyvést připájením vývodů k nožičkám procesoru, jak je například uvedeno na [16]. Toto řešení ovšem postrádá na estetičnosti a hrozí porušení CPU či jeho funkce, proto tato možnost bude vynechána. Zbývá tedy jeden port USB (druhý bude použit pro načítání systému, jak bude uvedeno dále) a 19 GPIO linek vyvedených na LED diody, tlačítko reset a také jako nepoužité piny procesoru. Z pohledu portu USB se jedná o poměrně velké možnosti, protože spousta dnešních přístrojů s rozhraním USB umí komunikovat a data by se tedy mohla načítat přímo skrze něj. Také může být připojen některý z převodníků USB/Serial a získáno tak plnohodnotné rozhraní RS-232. Navíc existují i další převodníky ze sběrnice USB na řadu jiných rozhraní, které mohou být použity například pro komunikaci s přístroji, jež budou sbírat data a skrze rozhraní USB budou tato data načítána do databáze. Ve volném portu USB se

tedy skýtají velké možnosti. Posledním volným rozhraním je 19 GPIO linek, které mohou také sloužit ke sběru dat. Je pouze nutné s jejich pomocí „emulovat“ některé standardní rozhraní nebo připojit specializovaný obvod (např. 8255 pro paralelní porty), který zprostředkuje chování svých výstupních portů kompatibilní s některým ze standardizovaných rozhraní. Nevýhodou využití „čistého“ GPIO rozhraní je možnost zničení některého z portů nebo dokonce celého procesoru připojením zařízení s jinými úrovněmi svých napětí na výstupu, než s jakými porty pracují. Stejným způsobem lze zničit porty, připojí-li se periferie s příliš vysokým odběrem proudu. Z těchto důvodů vznikla potřeba vstupně/výstupní desky pro připojení periférií.

4.1.3 A/D Převodník

Aby bylo možné otestovat schopnost vytvořeného embedded systému sbírat data, tak je zapotřebí jako jednu z jeho periférií připojit zařízení či obvod, který bude data poskytovat. V konkrétním případě byla zvolena možnost sbírání dat z A/D převodníku. Jelikož převodníků existuje celá řada, tak je nutné zvolit některý z vhodných zástupců pro dané embedded zařízení. Možnými zástupci na toto místo se zabývá následující tabulka:

Převodník:	MCP 3202	ADC0804CN	ADC0831CCN	AD7997BRUZ-0
Výrobce	Microchip	National Seminconductor	National Seminconductor	Analog Devices
Sběrnice	SPI	GPIO	GPIO	I2C
Rozlišení:	12bit	8bit	8bit	10bit
Napájení:	2,7 – 5,5V	6,5V	6,5V	2,7 - 5,5V
Cena (cca):	70,-	50,-	50,-	110,-

Tabulka 3 Vhodné A/D převodníky

Hlavním aspektem pro volbu se stala hlavně cena, která by u ryze demonstračního obvodu neměla být příliš vysoká. Jako nejvhodnější kandidát byl tedy vybrán převodník MCP 3202 i pro jeho možnosti nasazení a to jak na rozhraní GPIO, tak i případně na sběrnici SPI. Jedná se o dvoukanálový 12-ti bitový A/D převodník u nějž lze navolit, jakým způsobem budou zapojeny vstupy. Pro

komunikaci s obvodem jsou zapotřebí 4 linky, přičemž třemi se obvod nastavuje a ovládá dle následující Tabulka 4 a z jedné jsou čtená data.

	CONFIG BITS		CHANNEL SELECTION		GND
	SGL/DIFF	ODD/SIGN	0	1	
SINGLE ENDED MODE	1	0	+		-
	1	1		+	-
PSEUDO-DIFFERENTIAL MODE	0	0	IN+	IN-	
	0	1	IN-	IN+	

Tabulka 4 Konfigurační bity a jejich význam [14]

4.1.4 Úprava desky routeru pro účely volných vstupů/výstupů

Samotný router je nutno také mírně modifikovat. Rozhraní SPI je totiž vyvedeno pouze na plošky desky plošných spojů a rozhraní GPIO není vyvedeno vůbec. Rovněž napájení pro uvedené periferie by mělo být šířeno z desky, aby všechny periferie byly na stejném potenciálu a nevznikaly tak rušivé proudy zemními smyčkami. Na rozhraní SPI je tedy nutné připájet některý ze standardních osmi-pinových konektorů PC, aby bylo možné vytvořit jednoduché rozebíratelné spojení. Linky GPIO jsou vyvedeny pouze na LED diody a proto bude nutné tyto diody odpájet a vyměnit taktéž za konektor. Jako poslední je zapotřebí vyvést napájení a to buďto přímo z napájecího konektoru nebo konektorů USB.

Všechna uvedená zařízení jsou pouhým nutným základem potřebným pro výrobu nového systému schopného sbírat data. Dalšími možnými vylepšeními by mohlo být rozšíření vstupních a výstupních portů pomocí registrů a také připojení periférií pro jednoduchou signalizaci – např. několik LED diod nebo i složitější zařízení jako klávesnice a displej či wifi síťová karta. S pomocí těchto periférií a vylepšení by bylo možno rozšířit a získat mnoho nových funkcionalit daného systému.

4.2 SOFTWARE EMBEDDED SYSTÉMU PRO SBĚR DAT

Na následujících řádcích bude rozebrána problematika výběru vhodného software pro potřeby embedded systému na sběr dat včetně operačního systému a také základní filosofie tvorby, inicializace a nastavení operačního systému GNU/Linux. V textu budou také nastíněny možné volby software pro realizaci požadovaných funkcí. Z těchto možností bude zvolena vždy jedna, která bude implementována na koncovém zařízení. Budou také uvedeny závěry, které vedly k aplikaci dané volby.

4.2.1 Volba operačního systému

Z textu uvedeného v kapitole 2, která se zabývala mimo jiné i operačními systémy pro embedded systémy jasně vyplývá, že pro potřeby daného systému na sběr dat je nejvhodnější operační systém GNU/Linux nebo BSD. Z drobného internetového průzkumu těchto systémů a jejich distribucí plyne, že pro konkrétní hardware je širší podpora v operačním systému GNU/Linux a proto bylo rozhodnuto pro nasazení tohoto systému.

Operační systém GNU/Linux je šířen v několika tzv. distribucích. Každá distribuce se vyznačuje tím, že má jinak nastavené vlastnosti systému, jinak upravené jádro Linux, nainstalované jiné programy, jiný balíčkovací systém atd. Odlišností je velká řada. Nicméně základ zůstává stále stejný, tím je Linuxové jádro a základní systémové utility plynoucí z projektu GNU. Distribucemi vhodnými pro embedded systémy se zabývala například [1]. Ze všech distribucí uvedených v této knize jsou pro zvolený hardware vhodné pouze Embedded Gentoo, μCLinux, OpenWrt a projekty Linux From Scratch a OpenEmbedded. Jelikož daný hardware byl již v minulosti zprovozněn na distribuci OpenWrt nebo na distribucích z ní odvozených (Midge a Squidge), tak bude pro koncovou implementaci použita tato. Ostatní distribuce a projekty nebyly na daný hardware přímo portovány a proto je jejich implementace mnohem složitější nicméně na příkladu OpenWrt (ani v OpenWrt není router oficiálně podporován) bude uveden i obecný postup aplikace jakékoliv jiné distribuce na prakticky jakýkoliv embedded systém.

4.2.2 Základní filosofie práce systému GNU/Linux po jeho startu a při jeho kompilaci

Následující část textu se zabývá základní filozofií tvorby, inicializace a nastavení systému GNU/Linux jejíž pochopení je velice důležité pro nasazení systému GNU/Linux na embedded systém. Bez jejího zvládnutí by nemohl vzniknout co možná nejmenší a nejlepší systém pro běh daného embedded zařízení.

Chováním systému GNU/Linux po zapnutí se zabývají práce [6] a [1] a na následujících řádcích budou uvedeny nejdůležitější poznatky z těchto prací.

Jakmile se zapne zařízení, tak se nejdříve inicializuje samotný hardware a následně si procesor začne načítat z pevné paměti program. Na samotném začátku této paměti se musí nacházet tzv. bootovací sektor. V tomto sektoru je umístěn zavaděč, respektive jeho první část. Tato první část je velice krátký program, který volá druhou část umístěnou v další části paměti a ta už se stará o natažení jádra (kernelu) operačního systému do paměti a jeho následné spuštění. Během spouštění jádra se spouštějí jednotlivé funkce jádra jako plánovač, paměťový manažer a další. Jakmile se jádro celé spustí, tak se spustí program init umístěn v */sbin/init*. Tento program má identifikační číslo (PID) rovno jedné a zastřešuje všechny ostatní programy či procesy – všechny procesy jsou potomky procesu init. Proces init se stará o kontrolu a připojení souborového systému pro zápis (při načítání jádra je připojen pouze pro čtení), zapnutí odkládacího prostoru (swapu), spuštění sítě, spuštění dalších programů a další. Program init si nejprve načte soubor */etc/inittab*, který mu říká co se má dělat. Jako první je většinou načten tzv. inicializační skript, který se postará o připojení souborového systému, zapnutí swapu a dalších. Po jeho vykonání se určí jaká úroveň běhu (runlevel) má být spuštěna. Pomocí úrovní běhu lze řídit, které programy mají být spuštěny za jakých situací. Například úroveň běhu rovná pěti znamená běh s grafickým prostředím při kterém bude zapotřebí spustit i X server, který se stará o obsluhu grafického režimu. Při úrovni běhu rovné třem naopak grafické prostředí spuštěno není a proto lze X server vynechat. O to, jaké programy se mají a nemají spouštět se starají další skripty umístěné například ve složkách /etc/rcX.d (v systému Debian), kde X je číslo úrovně běhu. O úrovních běhu a kontrole jednotlivých sužeb by toho mohlo být napsáno spoustu nicméně pro

základní pochopení činnosti systému toto stačí a pro hlubší studium lze doporučit [1]. Během inicializace také bývá spuštěna jedna nebo více virtuálních konzolí pro přihlášení a práci v systému. Jakmile je tedy celá inicializace provedena, tak se lze přihlásit do systému a začít s ním pracovat.

Tímto byla ve zkratce rozebrána činnost systému po startu, která je velice důležitá pro pochopení neboť její úpravou lze značně urychlit start systému a také zredukovat množství programů spuštěných ihned po zapnutí systému. Tímto lze ušetřit cenné kB až MB operační paměti. Znalost této filozofie je také velice důležitá při vytváření celého systému bez využití některé z distribucí a tvoření operačního systému GNU/Linux od úplného začátku.

Při nasazování operačního systému GNU/Linux na jakékoliv zařízení se lze vydat dvěmi cestami. První z nich je kompilace systému na dané platformě v již běžícím systému. Tato cesta je mnohem jednodušší, nicméně málokdy na dané platformě již běží systém. Jestliže už běží, tak by bylo zřejmě jednodušší jej upravit než vytvářet nový. Tato cesta bývá využívána víceméně pouze na PC a tedy architektuře x86 nebo x86-64, kde je dostatečný výkon a také dostupnost všech prostředků pro kompilaci a přípravu nového systému. Příkladem tvorby takového systému může být postup uvedený v [7], kde je návod na opravdu minimální systém, který obsahuje jen ten nejnutnější základ. Druhým příkladem může být [18], kde je již rozebrána tvorba komplexnějšího systému. V případě embedded systémů bývá málokdy k dispozici kompletní vývojové prostředí nebo minimálně všechny prostředky k jeho vytvoření v daném systému. Dalším problémem je nedostatečný výkon embedded zařízení, který kompilaci nového systému může natáhnout až na několik hodin. Z těchto důvodů se používá druhá cesta, kterou je tzv. křížová kompilace (cross - compiling). Popis vytváření systému pomocí křížové kompilace je uveden například v [9], kde je uveden postup pro vytvoření systému na několika možných architekturách. Nevýhodou tohoto návodu je jeho neodladěnost. Návod je vhodný pro pochopení principu tvorby nového systému pomocí křížové kompilace, nicméně jej nelze doporučit k vlastní tvorbě nového systému (důvody budou rozebrány níže). Základem při vytváření nového systému je tzv. „toolchain“. Jedná se o prostředí ve kterém je nový systém vytvářen a kompilován. Toto prostředí je

vytvořeno na již běžícím systému a lze je vytvořit téměř pro jakoukoliv architekturu. Nejčastěji je k tvorbě toolchain zvolen PC převážně díky jeho výkonu a pohodlnosti práce. Nicméně nic nebrání tomu vytvářet toolchain například pro architekturu MIPS například na tabletu s architekturou ARM. Důležitá je pouze dostupnost prostředků pro vytváření toolchain – hlavně kompiler. Proces tvorby toolchain se skládá z následujících kroků:

- Nastavení proměnných prostředí
- Kompilace a instalace Binutils
- Nastavení, kompilace a instalace hlaviček zdrojů jádra
- Kompilace a instalace statického GCC
- Kompilace a instalace knihovny μClibc
- Instalace a kompilace kompileru GCC

V prvním kroku se nastaví proměnné prostředí na hodnoty požadované pro kompilaci budoucího systému. Těmito proměnnými jsou například cesty, kde se budou hledat patřičné programy, dále architektura hostujícího systému a nově vytvářeného a další. Také se v této části přípravy toolchain kompilují a instalují nástroje, které jsou zapotřebí pro tvorbu a správnou kompilaci toolchain. Těmito nástroji jsou například knihovny mpfr a gmp. Uvedené knihovny se starají o operace s pohyblivou řadovou čárkou a obecně o operace s čísly a jejich správné zaokrouhlování. V druhém kroku se potom připraví, zkompilují a nainstalují utility binutils. Jedná se o linker, assembler a další programy potřebné ke kompilaci zdrojových kódů do kódu strojového. Přípravou na kompilaci daného programu se rozumí stažení jeho zdrojových kódů, aplikace patchů a konfigurace. Kompilace samotná už probíhá pouze spuštěním příkazu make, který zpracovává a kompiluje celý balík nebo program pomocí skriptu či souboru nazvaného makefile. Instalací je pouhé zkopírování zkompilovaných programů a dalších potřebných souborů (např. dokumentace) na patřičná místa v adresářové struktuře budoucího systému. Po instalaci binutils je zapotřebí zkompilovat a nainstalovat do toolchain hlavičkové soubory jádra dané architektury. V těchto souborech jsou obsaženy definice všech funkcí, kterými jádro disponuje. Jakmile jsou nainstalovány, tak slouží pro získání informace o tom s jakými funkcemi se může pracovat a jakým způsobem. Tyto

informace potom slouží pro správnou kompilaci programů, jako jsou kompiler a základní knihovna rutin jazyka C obsažená v systému. Kompiler GCC se kompiluje na dvakrát. Poprvé je zkompilován jako statický (pouze jazyk C) kompilerem v hostujícím systému a využívajícím pouze hlavičky linuxového jádra. Jakmile je takto vytvořen, tak je ihned využit ke kompilaci základní knihovny jazyka C pro nový systém. S touto knihovnou už poté lze zkompilovat kompiler podruhé ovšem už i pro další jazyky. Takto vytvořený kompiler už potom slouží pro kompilaci programů do nového systému.

Je-li jednou vytvořena toolchain, je možno přejít k dalšímu kroku tvorby nového systému. Tímto krokem je kompilace a instalace programů, které na systému poběží. Před jejich vytvořením je ovšem potřeba ještě vytvořit adresářovou strukturu a to alespoň základní dle POSIX standardu. Této adresářové struktuře se říká RootFs nebo-li root file system nebo česky kořenový souborový systém. Tvorba jednoduché adresářové struktury dle výše uvedeného standardu je uvedená například v [7]. Jakmile je RootFs vytvořen, tak lze přejít ke kompilaci jádra. Jádro je možno použít stejné, jako při tvorbě toolchain. Na toto jádro je zapotřebí před samotným začátkem kompilace aplikovat patch nebo patche, které upravují jádro pro daný procesor a hardware. Pro dále vytvářený embedded systém jsou například aplikovány patche pro samotný procesor, patch pro funkci USB portů, patch na ovládání GPIO portů a další. Tyto patche je buďto nutné přímo vyrobit, což vyžaduje hluboké znalosti systému Linux nebo je získat přímo od výrobce (v dále uvedeném případě jsou použity patche výrobce procesoru - ADMtek) nebo je také možno aplikovat patche některé z distribucí (například patche od tvůrců OpenWrt) či od fanoušků zpřístupněné na síti Internet. Opatchované jádro je nutné nakonfigurovat například pomocí příkazu make menuconfig. Konfigurace probíhá formou nastavování či rušení vlastností jádra v grafickém nebo pseudografickém prostředí. Uvedená konfigurace pouze upravuje soubor .config. Proto by bylo možné pro konfiguraci použít i textový editor a měnit jednotlivé položky v tomto souboru. Nicméně touto cestou nelze získat některé skryté volby. I přes tuto nevýhodu může být přímá editace souboru .config v některých situacích výhodná. Velice výhodná je například pro určitý druh automatizovaného skriptu, který jádro přednastaví a klasickou

konfiguraci už se upraví pouze detaily. Správnou konfigurací jádra lze získat opravdu minimální a rychlý systém. Naopak špatnou konfigurací lze vypnout některé potřebné funkcionality nebo deaktivovat určité části hardware. Je proto zapotřebí věnovat konfiguraci zvýšenou pozornost, využívat nápovědu k jednotlivým položkám a v případě nejistoty raději danou položku ponechat ve výchozím stavu. Rozbor jednotlivých položek jádra by vydal na celou knihu a proto je nad rámec této práce. Po správné konfiguraci je jádro již možné zkompilovat pomocí výše vytvořené toolchain. Kompilací jádra vznikne obraz, který je poté spouštěn zavaděčem. V dalším je zapotřebí nainstalovat na systém základní systémové programy. Tyto mohou být doinstalovány dvojím způsobem. Buďto jednotlivě nebo pomocí jednoho komplexního balíku nazvaného busybox speciálně určeného pro embedded zařízení. I přestože v sobě zahrnuje velké množství programů a nástrojů, tak neobsahuje všechny jejich funkcionality a proto má mnohem menší velikost než jednotlivé programy. Jakmile jsou nainstalovány základní programy, tak je nutné do systému dodat skripty jako výše rozebraný init, hotplug, dále také fstab pro tzv. mount nebo česky připojování jednotlivých disků a řadu dalších. Pro popis jejich sestavení a funkce lze doporučit [9], kde jsou poměrně dobře rozebrány. Tímto je získán základní plně funkční systém do nějž už zbývají doinstalovat pouze speciální programy a jejich závislosti. Těmito programy je získána realizace požadovaných funkcionalit embedded systému a mohou jimi být například SSH server, knihovna pro kompresi souborů a řada dalších. Po doinstalování všech programů může být výsledný vytvořený RootFs nahrán do embedded systému. To, jakým způsobem bude RootFs nahrán do systému záleží na paměti z níž se tento systém načítá. Například pro pevný disk může být zkopírován ve stejném stavu v jakém byl vytvořen. Pro paměť flash instalovanou na embedded systému už bude zapotřebí prvně z RootFs vytvořit binární obraz, který může být nahrán do paměti. Další často využívanou možností je použít některý z komprimačních souborových systémů jakým je například squashfs. RootFs poté funguje na některém z uvedených souborových systémů, ale obraz takového systému je mnohem menší než bez využití těchto souborových systémů. RootFs nemusí být vždy instalován přímo na embedded systém, stačí pouze obraz jádra Linuxu, ve kterém je nastavena cesta k RootFs.

Tímto lze spustit GNU/Linux i na zařízení jež nedisponuje dostatečně velkou pevnou pamětí pro celý RootFs (tato alternativa bude také použita a blíže rozebrána níže). Embedded systém lze dokonce spustit i z paměti, do které lze nahrát pouze zavaděč systému. Jádro systému a RootFs jsou potom načítány ze vzdáleného umístění po síti. Tímto způsobem může vzniknout decentralizovaný systém obsahující jeden server a několik jednoduchých embedded systémů.

Uvedená podkapitola rozebrala způsob práce systému GNU/Linux ihned po zapnutí. Tato část byla inspirována [1] a [6]. Jedná se o část filozofie GNU/Linuxu, jejíž pochopení je velice důležité při vytváření systému zcela s „čistým štítem“. V druhé části jsou již rozebrány možné způsoby tvorby systému GNU/Linux a to hlavně tvorba tzv. toolchain, jež je zřejmě nejdůležitějším prvkem při výrobě nového systému pomocí křížové kompilace. V prostředí toolchain je totiž kompilován celý nový systém.

4.2.3 Výběr aplikačního software

Jelikož i interpretovaných jazyků, HTTP serverů, SSH serverů a databází pro embedded systémy existuje větší množství, je záhodno zvolit tu nejvhodnější variantu vzhledem k daným okolnostem. Ne vždy je ovšem software dostupný pro všechny architektury, což jej znemožňuje na těchto architekturách použít. Další překážkou může být již nainstalovaný či zvolený software, který nedovoluje nasazení toho či onoho dalšího softwarového produktu. Všechny aspekty zvoleného software je třeba předem důkladně zvážit tak, aby bylo splněno pravidlo tzv. occamova ostří. Tedy sestrojit nástroje dohromady tak, aby byly vzájemně co možná nejsladěnější a jejich instalace co nejjednodušší. Nemá smysl instalovat software tak, aby byl co možná nejmenší, když mezi sebou potom nebude správně spolupracovat a nebo jeho nastavení a zprovoznění zabere dlouhé hodiny.

4.2.3.1 Interpretovaný jazyk

Interpretovaných (skriptovacích) jazyků existuje poměrně široká škála. Mezi nejvýznamnější hráče na tomto poli na systému GNU/Linux bezesporu patří následující:

- Bash

- Perl
- Java
- Python

Každý z uvedených jazyků má své nesporné výhody i nevýhody a každý se většinou používá k jiným účelům. Jazyk Java je s výhodou využíván ve webových aplikacích. Naproti tomu jazyky Bash, Python a Perl jsou spojovány se systémovou administrací UNIXových systémů. Nevýhodou jazyka Bash je jeho závislost na dalších programech, což může způsobovat nekompatibilitu mezi jednotlivými verzemi. Výhodou potom je jeho implementace již v základním systému. Občas se ovšem najde i nějaký jiný shell (ksh, ash a další) implementovaný místo něj. Jazyky Python a Perl jsou velice silnými a mocnými a každý z nich dokáže vytvářet nejrůznější aplikace. Výhodou jazyka Python je oproti Perlu jeho implicitní objektová orientace, větší přehlednost kódu, díky pravidlům syntaxe pro jeho tvorbu a menší volností programátora a nakonec také jeho širší možnosti spolupráce s dalšími programovacími jazyky. Jeho jedinou nevýhodou je potom nepatrně větší velikost. Pro uvedené výhody bude nasazen právě tento interpretovaný jazyk. Na závěr části o skriptovacích jazycích dlužno dodat, že až na jazyk Java lze všechny uvedené jazyky na systému OpenWrt nainstalovat a to také na zvolené architektuře MIPS. Lze se tedy vydat i cestou s jazykem Perl nebo Bash.

4.2.3.2 HTTP Server

Příkladů HTTP (web) serverů vhodných na embedded systém je poměrně velké množství. Nicméně na systému OpenWrt jich už tak široká škála není, i když je stále z čeho vybírat. Ostatní lze sice doinstalovat kompilací přímo na novém systému nebo v toolchain, ale tato cesta je mnohem komplikovanější, než instalace pomocí správce balíčků opkg obsaženém uvnitř systému OpenWrt. Navíc kompilace nového software může být značně zdlouhavá. Zdlouhavost tohoto procesu je dána tím, že je zapotřebí vyřešit všechny závislosti daného programu a také nutností odstranění všech chybových hlášení, která ač by neměla, tak se prakticky pokaždé během kompilace objeví. Vzhledem k samotnému úvodu, kde bylo řečeno, že instalace by měl být co nejjednodušší, bude zvolen některý z HTTP serverů obsažených a

podporovaných přímo distribucí OpenWrt. Web servery vhodné pro zvolenou architekturu včetně základních parametrů jsou následující:

Název	Poznámka
Hiawatha	velikost 53kB, široká podpora protokolů včetně FastCGI, IPv6, SSL
Appweb	velikost 618kB, spotřeba RAM 800kB, podpora více jader a vláken, SSL a CGI
Axhttpd	velikost 7kB, spotřeba RAM 50-60kB, široká podpora šifrování a SSL, bez CGI
Mini-httpd	velikost 21kB, podpora CGI, SSL
Lighttpd	velikost 84kB, podpora FastCGI, Web 2.0, vysoké zatížení

Tabulka 5 HTTP Servery

Z uvedených serverů bude vzhledem k podpoře jazyka CGI, téměř nejmenší velikosti a malé spotřebě paměti RAM zvolen mini-httpd server. Hlavním atributem výběru byla velikost, nicméně podpora CGI, alespoň v základní verzi je také zapotřebí vzhledem ke komunikaci s jazykem Python.

4.2.3.3 Embedded databáze

Embedded databáze jsou většinou implementovány jako samostatný soubor, místo běžícího serveru v paměti. Šetří se tak procesorový čas, ale hlavně paměť RAM. Cenou za tuto výhodu je potom rychlost, která je o dost nižší. Na poli embedded databází již není tak široký výběr jako například v případě HTTP serverů. To alespoň z pohledu systému OpenWrt a architektury MIPS. Na samotném systému OpenWrt se lze setkat prakticky pouze s databázemi MySQL, PostgreeSQL a SQLite. Dalším limitačním faktorem by mohla být podpora jazyka Python, ale z uvedené trojice jsou podporovány všechny. Posledním atributem výběru tedy byla velikost v paměti RAM a spotřeba systémových prostředků. Jelikož PostgreeSQL a MySQL fungují jako servery v paměti a na daném systému je paměti minimum, tak byla zvolena databáze SQLite.

4.2.3.4 SSH Server

Také v případě SSH serverů není na systému OpenWrt příliš z čeho vybírat, nicméně varianty openssh a dropbear nejsou nikterak špatnými zástupci SSH serverů a jelikož byl opět kladen důraz na velikost a spotřebu paměti RAM, byl zvolen SSH server dropbear.

4.2.3.5 Výběr a popis souborových systémů

Souborových systémů je pro systémy GNU/Linux poměrně velké množství. Každý z těchto systémů se většinou specializuje na něco jiného. Zatímco některé se specializují spíše na klasický desktopový obsah, jiné se zase specializují na obsah serverových disků. Některé souborové systémy jsou vhodné pro práci s pamětí flash, jiné potom například pro CD-ROM. Ze všech systémů zde budou uvedeny alespoň ty nejznámější a také nejvhodnější pro embedded systémy.

Souborový systém ReiserFs byl jako jeden z prvních žurnálovacím souborovým systémem. Měl řadu výhod a na poli desktopů se těšil velké oblibě, ale po aféře jeho zakladatele v době vývoje verze ReiserFs4 jej ve velké míře začal předhánět souborový systém Ext4.

Systém Ext4 je po úspěšných verzích Ext2 a Ext3 dalším úspěšným systémem této řady. Stejně jako v případě ReiserFs se jedná o žurnálovací systém, tedy systém s bezpečným ukládáním dat. V současné době se jedná na poli desktopových PC zřejmě o nejlepší volbu. Na poli embedded systémů už ovšem většina jeho výhod, jako podpora velkých disků atd. moc veliký přínos nepřináší a proto je lepší u embedded používat systém Ext3. USB disk je tedy formátován systémem Ext3. Výhoda žurnálování se zvláště v případě zařízení, které nedisponuje vypínačem, pouze možností odpojení od napájení určitě hodí.

Dalším systémem se kterým je možno se setkat na poli embedded zařízení je jffs2. Tento souborový systém se používá převážně pro paměti typu NOR FLASH. Obraz jádra systému kopírovaném na FLASH paměť desky routeru běží na systému jffs2. Tato volba ovšem nebyla na uživateli, ale je dána systémem OpenWrt.

Jako poslední a neméně zajímavý systém je systém SquashFs. Jedná se o komprimovaný souborový systém, který nezabírá tolik paměťového prostoru.

V prvních verzích byl tento systém komprimován metodou gzip. V dnešní době se, ale používá téměř výhradně už jen komprimační systém LZMA. Tento systém se s výhodou využívá na discích CD, například v případě Live distribucí, ale také na embedded systémech pro zmenšení objemu dat. I v případě systému OpenWrt je celý systém komprimován právě tímto systémem.

Tímto je uzavřena kapitola voleb jednotlivých součástí nového systému a nyní lze přejít k realizaci. Stejně jako v předchozích kapitolách i kapitola voleb je „kuchařkou“ tvorby nového embedded systému a lze podle ní navrhnout hned několik různých embedded systémů nejen pro sběr dat.

5. ZPROVOZNĚNÍ A LADĚNÍ EMBEDDED SYSTÉMU PRO SBĚR DAT

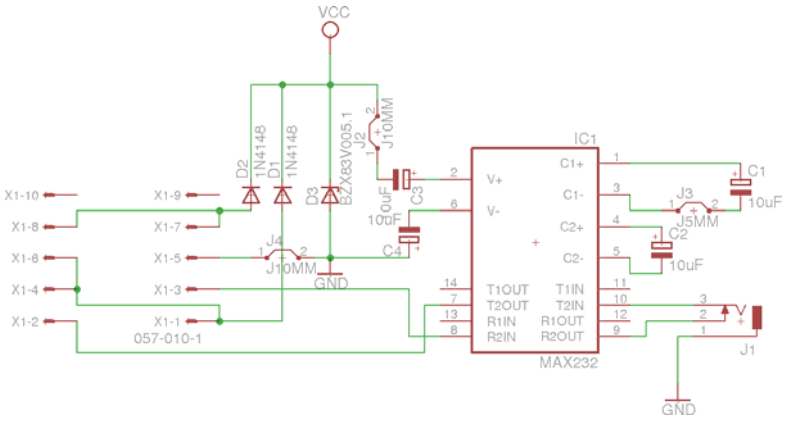
Následující kapitola popisuje jakým způsobem upravit hardware, aby bylo možné do routeru nahrát nový software. Dále realizaci rozhraní, přes které bude možno komunikovat s A/D převodníkem nebo také ovládat další periferie. Větší část kapitoly bude potom věnována tvorbě software, resp. způsobu kompilace a vytváření jádra OS a nakonec instalace, programování a nastavení programů zajišťujících vyžadované funkce systému.

5.1 TVORBA, ZPROVOZNĚNÍ A TESTOVÁNÍ HARDWAROVÝCH ROZŠÍŘENÍ

Následující podkapitola se zaobírá hardwarovými úpravami avizovanými v jedné z předchozích kapitol. Nachází se zde jejich návrh, ale i popis jednotlivých částí a součástí. Na závěr je vždy uveden také postup, jakým lze danou hardwarovou součást otestovat.

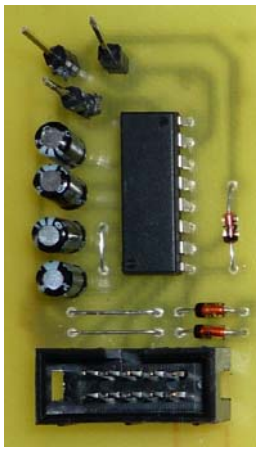
5.1.1 Převodník PC/Router

Celý převodník lze realizovat prakticky základním zapojením obvodu MAX232. Tento obvod je nábojovou pumpou, která se stará o změnu napětí z nižšího na vyšší o správných úrovních (směr deska -> PC) a dále také jistým druhem jednoduchého DC-DC měniče, který mění napětí z vyššího na nižší o správných úrovních (PC -> deska). Na rozdíl od základního zapojení je použito ještě malé zapojení s diodami, které se stará o vytvoření napájecího napětí pro obvod MAX232. Výsledné zapojení je potom uvedeno na následujícím obrázku:



Obrázek 2 Schéma zapojení převodníku PC/Router

Deska plošných spojů tohoto schématu v měřítku 1:1 je uvedena v příloze a na přiloženém DVD. Osazovací plánek desky je rovněž součást přílohy. Na místo vstupního portu RS-232 je použit pouze jeden ze standardních konektorů základních desek osobního počítače. To je z toho důvodu, že u dnešních základních desek PC již nejsou vyváděna rozhraní RS-232 ven z počítače, ale pouze uvnitř stejným druhem konektoru. V případě zájmu lze poté rozhraní RS-232 vyvést pomocí tzv. bracketu nebo česky záslepky ven. V tomto případě bude ovšem jednodušší použít malý kabel, který bude spojovat přímo PC s deskou převodníku. Stejný převodník může sloužit i ke komunikaci například se staršími typy mobilních telefonů, proto je zakončen konektorem typu JACK. Díky tohoto je možné připojit i jiná zařízení, která budou obsahovat pouze redukci z konektoru JACK na konektor daného zařízení. Výsledný převodník je uveden na následujícím obrázku:

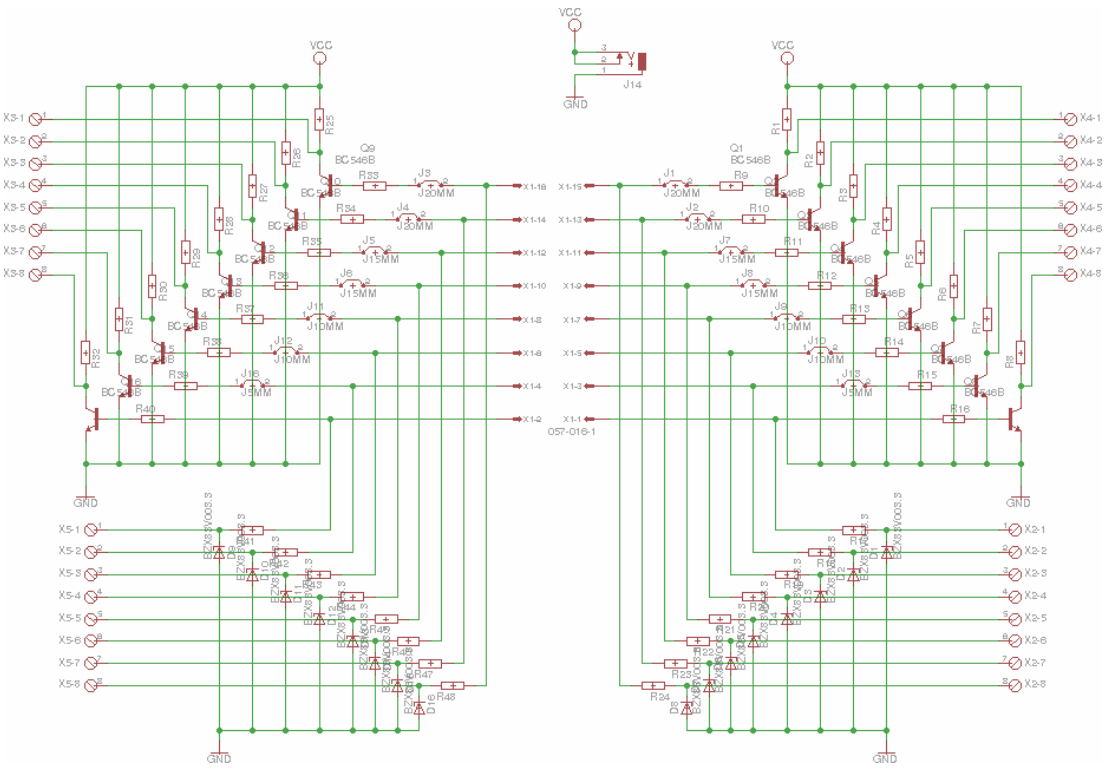


Obrázek 3 Převodník PC/Router

K oživení převodníku není zapotřebí žádných složitých procedur, pouze jej připojit mezi PC a zařízení, jež umí komunikovat pomocí sériové linky. Jakmile jsou obě zařízení fyzicky spojena, tak je zapotřebí správně nastavit parametry komunikace a poté už lze mezi zařízeními bez problému komunikovat. Pro test lze použit například některý ze starších mobilních telefonů a vyslat do něj AT příkaz: „AT+CGSN<CR>“, kde řetězec <CR> znamená speciální znak návratu na začátek řádku – ve většině programovacích jazyků reprezentován speciálním kódem: „\r”. Odpovědí mobilního telefonu by měl být kód „<OK>“, který označuje úspěšné přijetí příkazu a za tímto kódem by mělo následovat identifikační číslo – tzv. IMEI.

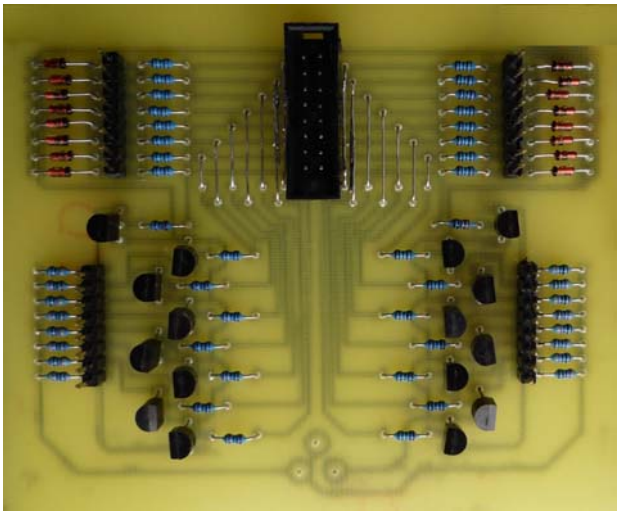
5.1.2 Vstupně – Výstupní deska pro připojení periférií

Jak již bylo avizováno, aby byly ochráněny vstupy a výstupy procesoru, tedy linky GPIO, je zapotřebí před tyto vstupy připojit nějaká ochranná zapojení. Jedním z těchto zapojení by mohlo být pomoci galvanického oddělení optočleny. Toto zapojení je, ale ovšem zbytečně drahé a postačí zapojení skromnější a to na následujícím obrázku:



Obrázek 4 Schéma vstupně-výstupní desky

V případě výstupů se vlastně jedná o vytvoření výstupu s otevřeným kolektorem pro GPIO linky. Díky tohoto typu výstupu zaniká možnost zničení GPIO linky připojením periferie se zvýšeným odběrem proudu nebo zkratem. Vstupy jsou potom ošetřeny pouze zapojením zenerovy diody a odporu. Zenerova dioda je chrání při připojení vyššího napětí, ale také přizpůsobuje pro komunikaci s periferiemi s úrovní TTL. Odpor se potom stará o omezení proudu do GPIO linek. Návrh desky plošných spojů včetně osazovacího plánku v měřítku 1:1 je uveden v příloze a na přiloženém DVD. Hotový převodník je uveden na následujícím obrázku:

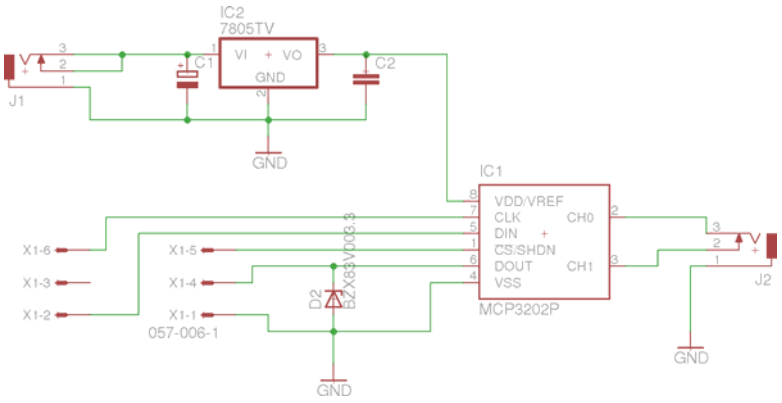


Obrázek 5 Vstupně-výstupní deska

Oživení periférní desky rovněž nevyžaduje žádné speciální zacházení, pouze připojení napětí. Deska by měla hned pracovat. Je potřeba si pouze dávat pozor na to, že výstupy s otevřeným kolektorem fungují prakticky jako logická negace a proto je zapotřebí všechny signály nastavovat na opačnou logickou hodnotu. Složitější situace je v případě testování. Zde je zapotřebí vytvářet umělý signál měnící svojí hodnotu mezi 0 V a 3.3 V pro výstupy z GPIO a měřit, zda –li dojde na výstupech tranzistorů ke změně úrovně. V případě testu vstupů je nutné připojit podobný signál pouze s úrovněmi 0 a 5 V a měřit, zda-li na patřičném vstupu pro GPIO dojde k omezení tohoto signálu na 3.3 V.

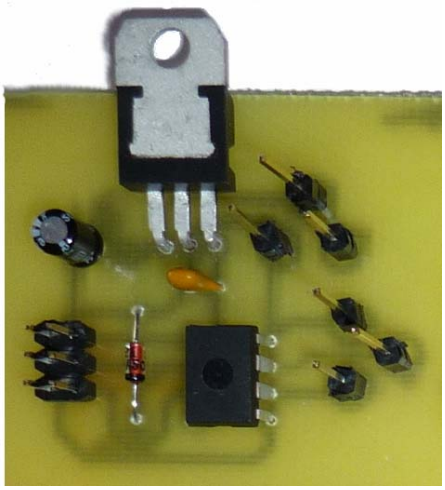
5.1.3 A/D převodník

Aby mohl být obvod A/D převodníku připojen na výše zmíněnou vstupně – výstupní periférní desku, tak je zapotřebí vytvořit malé zapojení, jež by toto umožňovalo. Toto zapojení je uvedeno na následujícím obrázku:



Obrázek 6 Schéma A/D převodníku

Jedná se o velice jednoduché zapojení, na kterém jsou k vidění pouze dva obvody. První z nich – stabilizátor 7805 se stará o napájení a vytvoření referenčního napětí pro druhý, kterým je již výše zmíněný A/D převodník. Celé schéma bylo jako předchozí dvě zapojení vytvořeno v návrhovém systému EAGLE, ve kterém rovněž vznikaly návrhy desek s plošnými spoji včetně desky pro A/D převodník. Deska je uvedená mezi přílohami v měřítku 1:1 včetně osazovacího plánku. Hotová a osazená deska je uvedena na následujícím obrázku:



Obrázek 7 A/D převodník

I oživení A/D převodníku vyžaduje pouhé připojení napájecího napětí. Pro účely testování je ovšem nutné zprostředkovat komunikaci s nějakým zařízením, např. PC. K tomuto účelu může skvěle posloužit převodník vytvořený v prvním bodě této podkapitoly. Na Rx signál převodníku bude připojen Dout signál A/D převodníku a na Tx potom bude připojen signál Din. Ostatní signály pro A/D převodník budou generovány z vnějšku a to včetně měřeného vstupního signálu. V počítači bude poté vytvořen testovací program sloužící pro otestování komunikace s A/D převodníkem. Testování by mohlo probíhat i jednodušší cestou a to pouhým připojením registru typu SIPO (serial-in, paralel-out neboli česky sério-paralelního registru), jak je například uvedeno v [20] a kompletní kontrolou všech linek pomocí programu. Tato cesta by také byla elegantnější z hlediska problémů se synchronizací, která se u předchozí verze testování téměř určitě projeví. Jelikož procesu není přidělován vždy stejný časový úsek, ale dle vytížení systému, tak se může občas stát, že některý z pulsů „vypadne“ a data dojdou neúplná nebo poškozená. Nicméně pro potřebu jednoho otestování je tato metoda naprosto dostačující.

V uvedené podkapitole byl uveden postup, jakým lze připravit periferie a další přípravky, které slouží ke komunikaci s routerem Edimax BR-6104KP a jeho nasazení jako embedded systému pro sběr dat. Tímto jsou vytvořeny a řádně otestovány všechny potřebné hardwarové komponenty a proto je možné se začít zaobírat jejich společným zprovozněním pomocí software.

5.2 PŘÍPRAVA A APLIKACE SOFTWARE PRO POTŘEBY EMBEDDED SYSTÉMU

Na následujících řádcích je uveden kompletní postup kompilace nového systému včetně aplikačního software. Celá kapitola se dělí na tři části. V první z nich je předpřipraven hostující systém. V druhém části je provedení kompilací nového systému a v poslední části potom příprava aplikačního software pro tento systém.

5.2.1 Nastavení a příprava hostujícího operačního systému pro kompilaci embedded operačního systému

Podkapitola nastavení a příprava hostujícího operačního systému pro kompilaci embedded operačního systému rozebírá kroky, které je nutné učinit pro korektní chod vývojového prostředí distribuce OpenWrt. Zabývá se tedy přípravou hostujícího systému.

Distribucí GNU/Linuxu pro PC je celá řada, nicméně autorem byla jako hostující systém, zvolena distribuce Ubuntu ve verzi 10.04 pro 64-bitové systémy. Tato distribuce byla zvolena pro její rychlou instalaci a poměrně dobře nastavený systém ihned po instalaci. Distribuce vychází z distribuce Debian, která je dle autora stabilnější a odladěnější, ale po její instalaci je zapotřebí spoustu programů doinstalovat a donastavit, což zpomaluje hlavní práci, kterou je tvorba nového systému. Distribuce Ubuntu byla nainstalována jako naprosto čistý systém, tedy bez jakýchkoliv změn (byla provedena pouze aktualizace) a na následujících řádcích budou rozebrány další kroky, které je třeba učinit pro správnou a bezproblémovou přípravu nového operačního systému. Jestliže budou doslovně dodrženy všechny uvedené kroky, lze pouhým zkopírováním toolchain z přiloženého DVD získat systém OpenWrt s nastavením s jakým byl vytvořen obraz pro embedded systém a lze tak kdykoliv opětovně vygenerovat daný obraz, či případně mírně pozměnit vlastnosti cílového systému. Všechny uvedené kroky lze aplikovat i s použitím jiné distribuce, pouze budou zapotřebí zřejmě jiné příkazy a jiné názvy téhož software. Bohužel ani tak nemusí být úspěch zaručen a proto je lepší použít zmíněnou distribuci nebo zopakovat všechny dále uvedené kroky. Celé počáteční nastavení systému je inspirováno návodem uvedeným ve zdroji [12], který je pouze ucelenějším popisem části dokumentace k projektu OpenWrt dostupné z domovské stránky projektu [11].

Nastavení je započato vytvořením nového adresáře s názvem openwrt ve složce */home*. Toto lze realizovat následujícím příkazem spuštěným v aplikaci terminál:

```
sudo mkdir /home/openwrt
```

Jelikož do adresáře home může zapisovat pouze administrátor, tak je zapotřebí příkaz spustit s administrátorskými právy (pomocí sudo). Aby nebylo zapotřebí pokaždé do složky přistupovat s administrátorskými právy, tak je vhodné této složce změnit vlastníka a to například následujícím způsobem:

```
sudo chown user /home/openwrt
```

, kde je položku user nutné nahradit aktuálně přihlášeným uživatelem. Tímto může ze složky uživatel číst, také do ní zapisovat a spouštět z ní soubory. V dalším kroku je zapotřebí do složky vstoupit pomocí následujícího příkazu:

```
cd /home/openwrt
```

Jakmile je složka připravena, lze přejít k dalšímu kroku, kterým je doinstalování aplikací potřebných pro práci s OpenWrt systémem. Instalaci lze provést následujícím příkazem:

```
sudo apt-get install quilt gawk flex zliblg-dev libncurses-dev
subversion g++
```

Během stahování a instalaci je uživatel dotázán zda-li chce opravdu nainstalovat uvedené balíčky. Toto lze s klidným svědomím odsouhlasit a pokračovat dalšími kroky. Chybí-li některý z uvedených balíčků v systému, tak je na tuto skutečnost uživatel upozorněn během konfigurace OpenWrt. V okamžiku doinstalování všech potřebných programů již lze přejít k získání systému OpenWrt a k jeho instalaci. Systém OpenWrt lze získat dvojím způsobem. Prvním způsobem je stažení archivu ze stránek projektu [11] a jeho následné rozbalení do výše vytvořené složky, druhým způsobem je stažení aktuální vývojové verze pomocí subversion klienta. V této práci byla zvolena druhá možnost, tedy stažení vývojové verze a to následujícím příkazem:

```
svn checkout svn: //svn.openwrt.org/openwrt/trunk data
```

Úspěšné stažení OpenWrt systému je signalizováno výpisem zakončeným textem podobným následujícímu:

```
Checked out revision 21195
```

Uvedená revize patří distribuci OpenWrt s názvem Backfire ve verzi 10.03. Po stažení systému se vytvoří ve složce openwrt nová složka data. Do této složky je zapotřebí vstoupit příkazem:

```
cd data
```

Jakmile je pracovním adresářem adresář data, tak je dále nutné stáhnout ještě nejnovější strom s aplikacemi, kterými lze distribuci OpenWrt rozšířit, příkazem:

```
./scripts/feeds update
```

Po dokončení operace je systém připraven na konfiguraci a následnou kompilaci zdrojových kódů distribuce OpenWrt. Aby bylo možné s routerem komunikovat pomocí sériové linky, je zapotřebí navíc doinstalovat terminál pro práci se sériovou linkou. Dle doporučení v [12] byl zvolen program minicom, který lze přiinstalovat do systému příkazem:

```
sudo apt-get install minicom lrzsz
```

Program lrzsz potom slouží pro nahrávání vytvořeného obrazu systému do desky routeru. Dokončením instalace všech programů uvedených výše je hostující systém již kompletně připraven na práci a vývoj operačního systému vycházejícího z distribuce OpenWrt. V následující kapitole je již rozebrána konfigurace a kompilace obrazu tohoto systému.

5.2.2 Konfigurace a kompilace operačního systému OpenWrt

Konfiguraci a kompilaci celého systému lze provést s pomocí dvou příkazů, jež budou uvedeny dále. První z nich spustí grafické prostředí konfigurace, ve kterém se nastavují jednotlivé volby cílového systému, druhým příkazem je již spuštěna kompilace po jejímž ukončení je k dispozici obraz systému. Na následujících řádcích budou uvedeny jednotlivé volby, jež byly změněny z výchozího nastavení. Popis všech voleb by byl nad rámec této práce, jelikož se jedná i o popis voleb samotného Linuxového jádra.

K samotné konfiguraci systému OpenWrt slouží příkaz:

```
make menuconfig
```

Tímto příkazem je spuštěno pseudo-grafické prostředí, ve kterém je možno nakonfigurovat celý systém OpenWrt a to včetně některých možností jádra. Na následujících řádcích bude vždy uvedena cesta k dané volbě (část stromové struktury konfiguračního prostředí) a vysvětlení jejího použití. Základ konfigurace je inspirován příspěvkem uvedeným na fóru OpenWrt [13].

```
Target System      --->
      (X) Infineon/ADMtek ADM 5120
```

Je zřejmé, že tento konfigurační parametr slouží k určení typu procesoru obsaženém v routeru.

```
Target Profile           --->
      (X) Edimax BR-6104KP (Unofficial)
```

I tuto volbu zřejmě netřeba vysvětlovat, jedná se totiž o nastavení profilu pro daný hardware.

```
Global build settings            --->
      [ ] Enable shadow password support
```

Podpora pro vytváření zahashovaného hesla v souboru */etc/shadow* není zapotřebí. Sníží se tím sice mírně bezpečnost, nicméně není předpokládáno, že by se jednalo o server se zvláště citlivými daty. V opačném případě by byla tato podpora vítána za cenu zvětšení velikosti jádra a tím i spotřebou paměti.

```
[*] Image configuration    --->
      (192.168.1.1) LAN DNS server
      (192.168.1.1) LAN Gateway
      (192.168.1.100) LAN IP Adress
```

Tato položka menu slouží k nastavení pro výchozí nastavení nového systému, obsahuje mimo jiné cesty ze kterých se budou číst programy, cestu pro program init a další. Výše uvedené volby slouží pro správné nastavení sítě. Uvedené nastavení je pro případ, že se bude cílové zařízení nacházet na síti s vlastním DNS serverem a výchozí bránou na IP adrese 192.168.1.1 a koncové zařízení nebude kolidovat se žádným jiným zařízením v síti s adresou 192.168.1.100. Toto nastavení lze později změnit také přímo v systému v souboru */etc/network.conf*.

```
Base system  --->
      < > dnsmasq
      <M> dropbear
      <M> firewall
      < > mtd
```

Zrušení výše uvedených položek nám zaručí dostatečně malý obraz systému, který se vejde do flash paměti integrované na desce routeru. Aplikace označené velkým *M* mohou být přiinstalovány později. Písmeno *M* značí modul, to znamená, že místo instalace do systému daného programu se pouze vytvoří balíček, který je poté možno doinstalovat. Úplné zrušení položky Mtd a Dnsmasq navíc zruší kompilaci a instalaci nepotřebných programů pro daný systém. V případě mtd se jedná o program, který je užitečný při upgradu z jiných firmwarů. V případě Dnsmasq se jedná o program sloužící jako nenáročný DNS a DHCP klient. Tento program by byl vhodný, jestliže by měl router i nadále sloužit také jako router.

```
Base System         --->
      <*> busybox         --->
            Configuration       --->
```

```
Coreutils           --->
      [*] stty
Login/Password Management Utilities    --->
      [*] addgroup
      [*] delgroup
         [*] Support for removing users from group
      [*] adduser
      [*] deluser
Miscelaneous Utilities    --->
      [ ] watchdog
```

Tyto volby umožňují během správy systému přidávat a mazat uživatele a skupiny. Volba programu stty umožňuje měnit nastavení sériové linky. Co se týká nastavení programu Busybox, tak v jeho případě je lepší žádnou volbu neubírat. Pouze v případě nedostatku místa na pevné paměti – v kořenovém souborovém systému. Funkcionalita watchdog slouží k monitorování a restartu „zamrzlého“ systému. Jelikož je ovšem tato kontrola systému vykoupena menším množstvím paměti RAM, jíž je na daném zařízení nedostatek, tak je watchdog odebrán.

```
Libraries    --->
      <M> libncurses
      <M> zlib
```

Knihovna ncurses slouží k obsluze terminálu a je vyžadována Pythonem, proto bude později doinstalována, jako balíček. Ze stejného důvodu jako Ncurses je přidána také knihovna zlib, její účel je ovšem implementovat kompresní metody pro komprimaci nebo archivaci souborů.

```
Network      --->
      <M> iptables
      < > ppp
```

Výše uvedenými volbami je dosaženo další úspory ve velikosti obrazu výsledného systému a také je odebrána podpora pro Point-To-Point protokol, který je využíván většinou pro připojení mezi internetovým zprostředkovatelem a speciálním koncovým zařízením – modemem. Není předpokládáno, že by měl daný embedded systém fungovat jako modem a proto je tato volba vynechána jako zbytečná.

```
Kernel modules      --->
      Block Devices       --->
            <*> kmod-scsi-core
            <*> kmod-scsi-generic
      Filesystems         --->
            <*> kmod-fs-ext3
      Other modules       --->
```

```
        < > kmod-button-hotplug
        < > kmod-input-gpio-buttons
        < > kmod-input-core
        < > kmod-ledtrig-adm5120-switch
        <*> kmod-gpio-dev
        <*> kmod-leds-gpio
    USB Support         --->
        <*> kmod-usb-storage
```

Tímto nastavením je řečeno, že v jádře bude zabudována podpora pro SCSI zařízení, jimiž jsou také USB disky. Další volbou je nastavena zapnutá podpora pro souborový systém Ext3 použitý na disku USB. V možnostech nastavení *Other modules* je vypnuta podpora pro tlačítka pomocí funkce hotplug. Tato funkce může sloužit pro připojení tlačítka reset do systému, nicméně není zapotřebí (reset může být vyvolán vzdáleně pomocí SSH) a proto také není instalována. Blikání LED diod podle přepínání switche uvnitř routeru také není zapotřebí a proto je také vypnuto. Zapnuta je funkce pro vytvoření nového zařízení uvnitř složky */dev*, jež bude umožňovat přímou komunikaci s GPIO rozhraním. Nakonec je také přidána funkce pro řízení LED připojených na GPIO, čímž se uvolní i LED, které jsou v systému původně určeny pro monitorování připojeného napětí a komunikace na sběrnici USB. Nastavením v položce USB Support se zapne podpora pro bloková zařízení připojená na sběrnici USB, tedy například USB flash disk.

```
Utilities           --->
    < >admswconfig
    <M> gpioctl
```

Vypnutím volby admswconfig je zrušena možnost ovládat interní přepínač CPU pro síťová zařízení. Jelikož ovšem není tato funkce zapotřebí, bude vypnuta. Kdyby bylo zapotřebí router i nadále využívat jako router, bylo by nutné tuto volbu ponechánu zapnutou. Utilita gpioctl slouží pro kontrolu a řízení linek GPIO a lze jí použít pro testování komunikace a linek GPIO.

Jsou-li všechny volby správně nastaveny dle výše uvedených, lze konfiguraci ukončit a uložit změny do konfiguračního souboru. Na uložení změn je uživatel dotázán při ukončování konfiguračního prostředí a je nutné zvolit Yes. V opačném případě jsou všechny změny nenávraně ztraceny. Dalším krokem v kompilaci nového systému je konfigurace jádra, jehož správnou konfigurací lze opět značně urychlit

běh a také spotřebu paměti RAM. V distribuci OpenWrt je konfigurační menu jádra vyvoláno příkazem:

```
make kernel_menuconfig
```

Konfigurační menu jádra opět nebude popisováno kompletně, ale pouze části, které byly změněny na rozdíl od základního nastavení.

První volbou je podpora desek:

```
Machine selection  --->
      ADM5120 Board selection   --->
            [ ] Cellvission CAS-771/771W support
            [ ] Cellvission NFS-101U/101WU support
            [ ] Compex NP27G support
            [ ] Compex NP28G support
            [ ] Compex WP54 familly support
            [ ] Edimax BR-6104K support
            [ ] Infineon EASY 5120-RT Reference Board support
            [ ] Infineon EASY 5120-WVoIP Reference Board support
            [ ] Infineon EASY 5120-ATA Reference Board support
            [ ] Infineon EASY 83000 Reference Board support
            [ ] Mikrotik RouterBOARD 111/112 support
            [ ] Mikrotik RouterBOARD 133 support
            [ ] Mikrotik RouterBOARD 133C support
            [ ] Mikrotik RouterBOARD 150 support
            [ ] Mikrotik RouterBOARD 153 support
            [ ] Mikrotik RouterBOARD 192 support
            [ ] Motorola Powerline MU Gateway
            [ ] OSBRiDGE 5GXi/5GXLi support
```

Tímto je odstraněna podpora většiny desek v seznamu. Označeny zůstanou pouze dvě desky firmy Edimax a to skutečná deska jež je obsažena v routeru a také deska routeru BR-6104WG/6114WG, která je svázána s USB podporou. Jejím odstraněním by došlo k vypnutí některých voleb konfigurace, mezi kterými jsou i důležité volby USB podpory, proto musí zůstat zatržena. Jedinou odznačenou deskou Edimax je BR-6104K, jež je sice identickou s konkrétním použitým typem, ale bez integrovaného USB rozhraní (je zapotřebí je přidat). Za zmíňku stojí také desky firmy Infineon z jejichž názvu jasně vyplývá, že se jedná o v samotném úvodu zmíněné vývojové desky.

```
General setup --->
      -*- Configure standard kernel features (for small systems) --->
            [ ] BUG() support
      [ ] Enable PCI quirk workarounds
```

Uvedenými volbami je dosaženo zmenšení jádra a také jeho spotřeby paměti RAM – je odstraněna podpora PCI chipsetů, jež nejsou na desce použity a také podpora pro signály BUG a WARN v jádře.

```
[*] Networking support    --->
      Networking options  --->
            < > 802.1Q VLAN Support
      [ ] Wireless --->
```

Odznačením výše uvedených voleb dojde k odstranění podpory nepotřebných součástí síťování a to konkrétně podpora bezdrátových sítí, jež není na uvedené desce implementována, ale mohla by se hodit v případě bezdrátového routeru a virtuální sítě VLAN, která zvyšuje efektivitu při přepínání a dalších síťových činnostech. Tuto funkci je vhodné ponechat v případě používání zařízení i jako routeru.

```
Device Drivers      --->
      <*> Memory Technology Device (MTD) support    --->
            [ ] Automatically set 'rootfs' partition to be root filesystem
            [ ] Automatically split 'rootfs' partitions for squashfs
            [ ] Automatically find and split TRX partitions
      SCSI device support --->
            {*} SCSI device support
            <*> SCSI disk support
            [ ] Probe all LUNs on each SCSI device
      < > Serial ATA (prod) and Parallel ATA (experimental) drivers    --->
      [*] Network device support--->
            [ ] Ethernet (1000 Mbit)  --->
            [ ] Wireless LAN    --->
      [ ] ISDN support    --->
      [ ] Watchdog Timer Support--->
      [ ] HID Devices     --->
      [*] USB support     --->
            <*> Support for Host-side USB
            <*> ADM5120 HCD support (EXPERIMENTAL)
            < > EHCI HCD (USB 2.0) support
            < > OHCI HCD support
            <*> USB Mass Storage support
            < > Datafab Compact Flash Reader support
            < > Freecom USB/ATAPI Bridge support
            < > USBAT/USBAT02-based storage support
            < > SanDisk SDDR-09 (and other SmartMedia, including DPCM)
support
            < > SanDisk SDDR-55 SmartMedia support
            < > Lexar Jumpshot Compact Flash Reader
            < > Olympus MAUSB-10/Fuji DPC-R1 support
```

```
        < > Support for Rio Karma music player
    [ ] Stagging drivers      --->
```

Všechna uvedená nastavení se týkají převážně odebírání nadbytečné podpory, jako například připojování všech jednotek v případě SCSI zařízení, jež se hodí pouze v případě speciálních zařízení jako jsou čtečky paměťových karet a podobných. V případě použití běžného USB flash disku je tato volba zbytečná. V případě síťování byla opět odstraněna podpora bezdrátových zařízení a také síťových rozhraní s přenosovou rychlostí 1Gb/s jelikož dané zařízení disponuje pouze síťovými kartami se 100Mb/s. Dále byla odstraněna podpora ISDN, která není v případě ethernetového zařízení zapotřebí. Podpora funkce watchdog a HID (human interface devices) byly rovněž odstraněny. Nakonec byla přidána podpora pro USB kontrolér použitý na desce a tzv. mass storage zařízení.

```
File systems --->
    <*> Ext3 journalling file system support
```

Tímto je zapnuta podpora pro systém souborů Ext3, který bude použit na externím disku.

```
Kernel hacking      --->
    Default kernel command string:
console=ttyS0,115200 root=/dev/sda1 init=/etc/preinit ro rootwait
```

Tento příkaz říká jádru, aby načítalo rootFs z připojeného disku.

Nyní lze přejít k samotné kompilaci příkazem:

```
    make
```

V tomto okamžiku je spuštěna kompilace celého systému. Jednotlivé kroky se zobrazují na terminálu a budou vysvětleny níže. Těmito kroky jsou:

```
make [1] world
make [2] tools/install
make [2] toolchain/install
    make [3] -C toolchain/binutils prepare
    make [3] -C toolchain/binutils compile
    make [3] -C toolchain/binutils install
    make [3] -C toolchain/gcc prepare
    make [3] -C toolchain/kernel-headers prepare
    make [3] -C toolchain/kernel-headers compile
    make [3] -C toolchain/kernel-headers install
    make [3] -C toolchain/uClibc prepare
    make [3] -C toolchain/gcc compile
    make [3] -C toolchain/uClibc compile
    make [3] -C toolchain/gcc install
    make [3] -C toolchain/uClibc install
```

```
make [2] target/compile
make [2] package/cleanup
make [2] package/compile
make [2] package/install
make [2] package/rootfs/prepare
make [2] target/install
make [2] package/index
```

Výše uvedený výpis je zkráceným nicméně dobře ilustruje co systém OpenWrt při kompilaci dělá. Na příkladu systému OpenWrt je dobře vidět principy zmíněné v podkapitole filozofie GNU/Linuxu. V prvním kroku jsou připraveny proměnné prostředí a nastaveno samotné prostředí. V druhém kroku jsou přiinstalovány nástroje potřebné pro korektní kompilaci systému OpenWrt. Mezi těmito nástroji lze vidět například gmp a mpfr zmíněné taktéž v podkapitole o filozofii linuxu. Že se jedná o nástroje, které vyžaduje i samotná automatická kompilace systému OpenWrt dokazuje například přítomnost programu Sed, který se využívá například v automatizovaných skriptech pro práci s řetězci a to ať už v souboru nebo ze vstupu do programu. Po nainstalování všech potřebných nástrojů se spouští tvorba toolchain, která jak je vidět téměř přesně kopíruje scénář uvedený ve filozofii GNU/Linuxu. Dále následuje kompilace jádra systému a následné čištění systému. Toto čištění nebylo zmíněno ve filozofii GNU/Linuxu nicméně se doporučuje. Jako další probíhá kompilace a instalace doprovodných programů. Po jejich instalaci je vytvořen RootFs a v něm také všechny potřebné skripty. V samotném závěru jsou do nového RootFs přesunuty zkompilované programy a jádro z výsledného systému je překonvertováno na obraz, který je možno nahrát přes sériovou linku, dále obraz pro nahrávání pomocí webového rozhraní a nakonec je také vytvořen RootFs jako soubor archivu tgz. Toto všechno lze nalézt ve složce */home/openwrt/data/bin/adm5120/*, kde se také nachází adresář s balíčky, které lze později doinstalovat.

Touto podkapitolou byl vytvořen zcela nový systém, který by měl mít minimální velikost a maximální efektivitu využití systémových prostředků, protože neobsahuje žádný přebytečný kód.

5.2.3 Příprava aplikačního software

V této podkapitole bude k jádru systému a základnímu softwaru přikompilován také doprovodný aplikační software, který bude realizovat funkce SSH serveru, HTTP serveru a databáze.

Před samotným začátkem kompilace je nutné updatovat OpenWrt kompilační systém, aby obsahoval všechny aktuálně dostupné balíčky a to příkazem:

```
./scripts/feeds update
```

Pomocí tohoto skriptu je možné do systému (konfiguračního menu) přidávat, ubírat, hledat nebo vypisovat všechny dostupné balíčky pro systém OpenWrt. Hledání například programu Python v seznamu balíčků lze realizovat příkazem:

```
./scripts/feeds search python
```

Podobným způsobem lze vyhledávat i v popisu balíčků. Hledá-li se například vše co má spojitost s HTTP (např. HTTP server), tak je potřeba použít následující příkaz:

```
./scripts/feeds search http
```

I když je výpis těchto příkazů poměrně dobrý, tak je vždy dosti obsáhlý. Proto je vhodné použít pro další filtraci ještě program grep, například následovně:

```
./scripts/feeds list | grep web
```

Tímto se z celého seznamu balíčků a jejich popisů vyberou pouze ty, které obsahují slovo web (nejčastěji web kamery a web servery). Pro instalaci všech zvolených balíčků aplikačního software do konfiguračního menu lze použít následující příkaz:

```
./scripts/feeds install python python-sqlite mini-httpd
```

Ostatní potřebné knihovny a programy, tzv. závislosti, se přidají automaticky (libffi, sqlite3). Nyní lze opět vyvolat konfigurační menu systému OpenWrt a doinstalovat potřebný aplikační software:

```
make menuconfig
```

V již známém menu se lze rovnou přesunout ke zvoleným balíčkům a přidat je jako moduly:

```
Libraries    --->
      database     --->
            <M> libsqlite3
      <M> libffi
Network      --->
      Web          --->
            <M> mini-httpd
Languages    --->
      Python       --->
```

```
<M> python
<M> python-mini
<M> python-sqlite
<M> python-sqlite3
```

Po správném označení všech uvedených balíčků je zapotřebí pouze ukončit konfigurační menu a uložit změny. Poté lze již opětovně zkompilovat systém, který již bude navíc obsahovat nové balíčky:

```
make
```

Tímto je získán již kompletní systém, který je už pouze zapotřebí nainstalovat na zařízení routeru a nastavit.

5.2.4 Ostatní distribuce

Uvedený způsob kompilace nového systému je velice podobný všem Linuxovým distribucím a lze se s ním setkat s většími či menšími rozdíly. Některé distribuce jako [9] se zabývají kompletně celou kompilací od nuly. Tento způsob vyžaduje velikou trpělivost, protože tvorba toolchain je, jak je napsáno v dokumentaci embedded gentoo tak trochu černým uměním. Další distribuce poskytují, podobně jako OpenWrt, jednodušší menu, které uživateli pomůže s nastavením systému a kompilací toolchain [10]. A některé distribuce již přímo poskytují zkompilované toolchain pro dané architektury. Jakmile je toolchain hotová, tak je již na půl vyhráno, protože v dalších krocích už je tvorba systému jednodušší. I když sestavení všech skriptů a kompletní nastavení celého systému do kýžené podoby také zabere několik dní času. Celá kompilace nového systému tedy lze shrnout do čtyřech po sobě jdoucích bodů:

- Tvorba toolchain
- Kompilace jádra + příprava RootFs
- Tvorba skriptů
- Instalace aplikačního software

S tímto postupem se lze setkat vždy při křížové kompilaci a záleží pouze na tvůrcích distribuce, jak jsou tyto body náročné a jaké přinášejí možnosti.

5.3 UVEDENÍ DO PROVOZU

Následující část textu nabízí postup nahrávání výše připraveného software do embedded systému, jeho nastavení a spuštění.

5.3.1 Instalace RootFS

V prvním kroku je nutné připravit USB flash disk pro nahrání kořenového souborového systému – RootFs. Jelikož byla zvolen souborový systém Ext3 a v jádře Linux zvolena podpora pouze pro tento systém, tak je nutné USB flash disk do tohoto souborového systému naformátovat. Toto lze provést dvěma způsoby. První z nich je pomocí grafického rozhraní programu *Diskový nástroj* (palimpsest) umístěný na systémové liště v nabídce *Systém* pod položkou *Správa*, druhý potom přímo v konzoli. Jelikož je disk automaticky připojen (namountován) ihned po zapojení do PC a takový disk nelze formátovat, tak je nutné jej prvně odpojit příkazem:

```
sudo umount /dev/sda1
```

nebo v případě použití programu *Diskový nástroj* jej lze odpojit pouhým kliknutím na *Odpojit svazek*. Příkaz sudo ve výše uvedeném příkazu může i nemusí být zapotřebí, to záleží pod kterým uživatelem se daný disk připojí, což v případě prvního připojení bývá root a proto je sudo zapotřebí. Nicméně odpojením s administrátorskými právy nelze nic pokazit. Řetězec */dev/sda1* reprezentuje cestu k danému zařízení v systému. Je-li do systému již připojen jiný disk se SCSI rozhraním, typicky S-ATA disk, tak USB flash disk bude až za tímto zařízením a cesta k němu bude tedy */dev/sdb1* atd.. Pozor, zařízení */dev/sda*, */dev/sdb* atd. reprezentují samotné zařízení, nikoliv paměťový prostor v tomto zařízení a proto se neformátují souborovým systémem. Výpis všech disků připojených do systému lze získat příkazem:

```
ls /dev | grep sd.1
```

Jedná se o výpis všech hardwarových zařízení v systému přesměrovaných pípou do programu Grep, který umí vyhledávat v řetězci jež je mu předán jako vstup, regulární výraz který je druhým vstupem (v tomto případě *sd.1*). Regulární výrazy jsou velice silným nástrojem při zpracování textu a jsou hojně využívány jazyky jako je Bash, Python, Perl a řadou dalších. Pro jejich studium lze doporučit dokumentaci jazyka Bash nebo některou z knih: [5], [4]. Samotný formát disku do systému Ext3 lze po odpojení provést následujícím příkazem:

```
sudo mkfs.ext3 /dev/sda1
```

, kde */dev/sda1* opět reprezentuje USB flash disk v systému. V grafickém prostředí programu *Diskový nástroj* toto lze provést kliknutím na tlačítko *Formátovat svazek* a nastavením patřičných voleb. Těmito volbami jsou souborový systém – Ext3 a případně také název disku. Položkou *Převzít vlastnictví souborového systému* je navíc získána možnost zápisu a čtení souborového systému pro aktuálního přihlášeného uživatele. Tato volba je velice vhodná pokud je plánována i další práce v grafickém prostředí. Jakmile je formátování ukončeno, tak musí být nově vytvořený souborový systém opět připojen. Bylo-li zařízení formátováno v konzoli, tak je nejprve nutné vytvořit adresář do kterého se bude zařízení připojovat a to například následovně:

```
sudo mkdir /mnt/disk
```

Cesta */mnt/disk* může být nahrazena jakoukoliv jinou, ale je zapotřebí si ji pamatovat do následujícího příkazu, kterým se disk připojí do systému:

```
sudo mount /dev/sda1 /mnt/disk
```

Totéž lze provést jedním kliknutím v grafickém prostředí programu *Diskový nástroj* na tlačítko *Připojit svazek*. Tímto je disk USB připraven na nový systém. Zkompilovaný systém umístěný ve složce */home/openwrt/data/bin/adm5120* pod názvem *openwrt-adm5120-router_le-rootfs.tgz* lze nyní přesunout na USB disk následujícím příkazem:

```
sudo cp bin/adm5120/openwrt-adm5120-router_le-rootfs.tgz /mnt/disk
```

Pouhým překopírováním, ale ještě práce není hotova, je totiž zapotřebí celý systém na disk rozbalit. Aby bylo možno soubor rozbalit, tak je zapotřebí se přepnout do adresáře disku:

```
cd /mnt/disk
```

a v tomto adresáři spustit následující příkaz pro rozbalení archivu:

```
sudo tar –xzf openwrt*.tgz
```

Hvězdičkou se v tomto zápisu automaticky nahradí zbytek názvu souboru. Bylo-li by ovšem v daném adresáři více souborů archívu tgz s názvem openwrt, tak by se rozbalily všechny uvedené. Archiv by bylo možné rozbalovat i původní uložený ve složce openwrt, ale pro zálohu je lepší jej mít přímo na USB disku. V případě nedostatku místa na USB disku by potom mohl být smazán. Aby bylo možné do systému doinstalovat i balíčky, které byly během kompilace OpenWrt vytvořeny, tak je zapotřebí je také překopírovat do RootFs a to následovně:

```
sudo cp –r /home/openwrt/data/bin/adm5120/packages /mnt/disk/packages
```

Nyní je systém připraven na nahrávání a první pokus o spuštění, čímž se zabývá následující podkapitola.

5.3.2 Nahrávání obrazu nového systému a první spuštění

Je-li systém souborů úspěšně vytvořen může se přejít k nahrávání vytvořeného obrazu nového systému do desky routeru. Nejprve je nutné připojit převodník PC-Router. Po správném připojení je za potřebí navázat spojení mezi těmito zařízeními. Pro tento účel poslouží program Minicom. Aby program Minicom pracoval správně tak je nutné jej prvně nastavit a to spuštěním dle následujícího příkazu:

```
minicom –s
```

Tímto je programu Minicom řečeno, že bude nastavován. V konfiguraci je zapotřebí zvolit *Nastavení sériového portu* a v následujícím okně nastavit správné sériové zařízení – sériový port standardně bývá na */dev/ttyS0*, přenosová rychlost je 115200Bd, stop bit je jeden, kontrola paritou žádná a kontrola toku není ani hardwarová, ani softwarová. Po správném nastavení se spustí program minicom připraven komunikovat se zařízením na sériovém portu. V takové chvíli je třeba připojit napájecí napětí do routeru. Proběhlo-li nastavení a připojení správně zobrazí se v konzoli následující oznámení zavaděče:

```
ADM5120 boot:
```

V tomto okamžiku je nutné třikrát rychle stlačit mezerník, aby byla aktivována následující výzva zavaděče:

```
Linux Loader Menu
=================
(a) Download vmlinuz to flash ...
(b) Download vmlinuz to sdram (for debug) ...
(c) Exit

Please enter your key :
```

Po stisku klávesy *A* se spustí další okno očekávající data k nahrávání:

```
Downloading.......
```

Nyní je nutné dostatečně rychle stisknout klávesy *Ctrl + A* a poté klávesu *Z*. Touto kombinací kláves je spuštěno dialogové okno programu Minicom, ve kterém lze mimo jiné stiskem klávesy *S* odeslat soubor. Ještě před odesláním souboru je uživatel

dotázán na protokol, jakým se mají data přenášet. Mezi možnostmi je i *xmodem*, který je nutné zvolit. Po tomto výběru následuje výzva uživateli na výběr souboru. V zobrazeném okně lze vybrat *[Goto]* a zadat cestu k obrazu jádra a to: */home/openwrt/data/bin/adm5120*. Tímto je zobrazen výpis daného adresáře, ve kterém už je pouze zapotřebí zvolit soubor *openwrt-adm5120-router_le-br-6104kp-squashfs-webui.bin* a stiskem klávesy *Enter* odeslat do routeru. Je-li uživatel dostatečně rychlý, je tímto započato zobrazení načítání odesílaných dat. Toto načítání je zobrazováno necelých 5 minut, po kterých je celý obraz nového systému (1535kB) nahrán do flash paměti routeru a v konzoli je zobrazena informace o průběhu nahrávání:

```
Downloading.........PASS
Eraseing nor flash......PASS
Programing nor flash...PASS
```

a opět výzva zavaděče pro nahrávání nového obrazu. V takovém okamžiku je nutný stisk klávesy *C* pro ukončení této výzvy. Po stisku klávesy je automaticky započato načítání nového systému. Před jeho úspěšným načtením, hlavně jeho RootFs části je ovšem nutno ještě router restartovat odpojením a připojením napájení. V dalším startu se již na výzvu zavaděče neodpovídá a nový systém je nastartován. Úspěšný start lze ověřit stiskem klávesy *Enter*. Touto akcí je vyvoláno „logo“ OpenWrt v následujícím tvaru:

```
 _______                     ________        __
|       |.-----.-----.-----.|  |  |  |.----.|  |_
|   -   ||  _  |  -__|     ||  |  |  ||   _||   _|
|_______||   __|_____|__|__||________||__|  |____|
         |__| W I R E L E S S   F R E E D O M
KAMIKAZE (bleeding edge, r17053) -------------------
 * 10 oz Vodka       Shake well with ice and strain
 * 10 oz Triple sec  mixture into 10 shot glasses.
 * 10 oz lime juice  Salute!
---------------------------------------------------
```

V tuto chvíli je uživatel přihlášen do systému a je mu dovoleno jej spravovat.

5.3.3 Správa nového systému a instalace aplikačního software

Pokračováním zprovoznění nového systému je jeho nastavení a instalace aplikačního software, kterým se zabývá text na následujících řádcích.

Jako první z balíčků je nejvhodnější zprovoznit SSH server, který umožňuje vzdálenou správu přes rozhraní ethernet, které je mnohem rychlejší a nevyžaduje již převodník. Na začátku je zapotřebí vstoupit do složky s balíčky:

```
cd /packages
```

Nyní lze pomocí následujícího příkazu vypsat všechny dostupné balíčky:

```
ls
```

Pro instalaci je, jak již bylo zmíněno výše, používán správce balíčků opkg. Tento správce je jednodušší verzí správce dpkg známého z distribuce Debian. Syntaxe ovládání je poměrně jednoduchá a správce samotný zvládá jen několik málo příkazů. Popis celého ovládání a všech možných voleb lze zjistit příkazem:

```
opkg help
```

V současné chvíli je zajímavý pouze příkaz install, kterým budou všechny balíčky instalovány. Nyní je vhodné definovat heslo pro uživatele root, které bude poté použito i pro připojování. Změnu (vytvoření) hesla lze vyvolat příkazem:

```
passwd
```

Tento příkaz vygeneruje výzvu pro zadání nového hesla a poté druhou výzvu pro zopakování hesla. Jestliže hesla souhlasí, tak je heslo pro uživatele root změněno.

Jak již bylo zmíněno, tak jako první bude instalován SSH server s názvem dropbear a to následovně:

```
opkg install dropbear*mipsel.ipk
```

Hvězdička v příkazu automatický nahradí zbytek názvu. Toto je vhodné v případě, že se na systému nenachází více verzí stejného balíčku. V opačném případě by se správce pokusil o instalaci všech těchto balíčků, což by téměř určitě nedopadlo dobře. Balíček by bylo možno instalovat také ze sítě Internet, ale tato volba není doporučena, jelikož v repozitáři OpenWrt se nacházejí balíčky, které byly kompilovány v systému s obecným nastavením. Může jim tedy nějaká závislost chybět, což může i nemusí znemožňovat jejich korektní práci. Během instalace balíčku dropbear jsou vypsány dvě chyby související s chybějícími soubory. Tyto chyby lze přejít bez povšimnutí, protože nemají na běh SSH serveru vliv.

Jakmile je tedy server nainstalován, tak je možné jej spustit a to následujícím způsobem:

```
/etc/init.d/dropbear start
```

Po startu SSH serveru je již možné vyzkoušet připojení na nový systém pomocí SSH protokolu z hostujícího počítače a to pomocí následujícího příkazu:

```
ssh root@192.168.1.100
```

IP adresa 192.168.1.100 je stejná, jako ta, která byla použita během konfigurace OpenWrt systému v předchozí kapitole. Nyní je uživatel upozorněn, že autentifikace serveru na této adrese nemůže být uskutečněna a zda-li chce uživatel skutečně pokračovat. Na tuto výzvu lze beze strachu odpovědět „yes“. Autentifikace totiž nemůže být uskutečněna, jelikož právě vytvořený server nemá RSA podpis schválen žádnou certifikační autoritou. Je-li výzva odsouhlasena bude uživatel dotázán na heslo. Po jeho správném doplnění se opět zobrazí logo OpenWrt, jako v případě prvního připojení do systému pomocí sériové linky. Nyní lze tedy sériovou linku odpojit a ukončit běh programu minicom. Dalším vhodným krokem je přidat link (odkaz) na spouštěcí skript přímo mezi úrovně běhu. Na systému je obsažená pouze jediná úroveň a lze se do ní přepnout následujícím příkazem:

```
cd /etc/rc.d
```

V okamžiku kdy se systém nachází v této složce, tak lze vytvořit link následovně:

```
ln –s ../init.d/dropbear S50dropbear
```

Tímto se bude SSH server pouštět vždy po zapnutí systému. Jelikož číslo 50 je poměrně vysoké, tak SSH server nebude spuštěn přímo po startu, ale až naběhnou další funkce, jako obsluha USB, síť a další. Test automatického spuštění lze provést restartem zařízení a opětovném navázání spojení po asi 15-ti vteřinách. Restart systému lze realizovat následujícím příkazem:

```
reboot
```

Tímto je systém již zcela připraven na komunikaci a konfiguraci pomocí SSH protokolu. Nyní lze přejít k instalaci Pythonu, tedy „řídícímu centru“ celého systému. Instalace Pythonu je velice podobná instalaci SSH serveru, pouze s tím rozdílem, že je nejprve potřeba nainstalovat závislosti a až poté je možné nainstalovat samotný Python. Instalaci závislostí lze provést příkazem:

```
opkg install zlib*mipsel.ipk librt*mipsel.ipk libpthread*mipsel.ipk
libsqlite3*mipsel.ipk libffi*mipsel.ipk
```

Hvězdičky v příkaze mají stejný smysl, jako v případě dropbearu, tedy zjednodušení zápisu. Tohoto lze také využít v případě tvorby automatického skriptu pro vytváření

celého systému. Takto vzniklý systém potom nebude závislý na určité verzi balíčku. Po nainstalování závislostí lze přejít k instalaci samotného Pythonu a to příkazem:

```
opkg install python*mipsel.ipk --force-overwrite
```

Tímto se nainstaluje nejen samotný Python, ale i všechna související rozšíření jazyka, jako třeba rozšíření pro SQLite databázi. Volba –force-overwrite je zde, aby zamezila chybě, která se zobrazí v případě instalace bez této volby. Problém je v tom, že si balíček Python a Python-mini přepisují data. Toto ovšem není problém, protože balíček Python-mini je jen zjednodušenou verzí čistého Pythonu, který jej ovšem v systému OpenWrt má jako jednu ze závislostí. Musí tedy být nainstalovány oba s tím, že mají povoleno si přepsat data. Lépe řečeno balíček Python přepíše data balíčku Python-mini. Toto je zřejmě chyba v systému OpenWrt než v samotné instalaci a je možné, že s další revizí bude již odstraněna. Test jazyka python lze provést jednoduchým skriptem vypisujícím známé „Hello World“. Jako editor lze použít buďto v systému již zabudovaný editor vi (je součástí balíčku busybox) nebo pro neznalé ovládání tohoto editoru je vhodnější editor nano pracující podobně jako poznámkový blok. Tento editor je, ale nutné doinstalovat podobným způsobem, jako Python (závislostí je pouze libncurses). Další možností je využití grafického prostředí zvaného idle přímo poskytovaném Pythonem a napsat skript v něm na hostujícím počítači (na embedded systému není nainstalován grafický server). Po úspěšném dokončení skriptu jej pak lze jednoduše zkopírovat pomocí příkazu scp – bude rozebráno dále. Po spuštění editoru jsou vloženy následující řádky:

```
#!/usr/bin/python
print ("Hello world!!!")
```

Editor nyní může být ukončen a skript uložen například jako hello.py. Syntaxi jazyka Python je nad rámec této práce a nebude nikde v textu podrobně popisována. Syntaxe jazyka Python je podrobně popsána v [5], odkud ji zná také autor tohoto textu a nebo na webových stránkách tohoto jazyka. Spustit takto vytvořený skript lze dvěma způsoby. Prvním z nich je zavolání interpretu, který jej spustí a to následovně:

```
python hello.py
```

Druhým způsobem je skriptu nastavit práva na spouštění a to příkazem:

```
chmod a+x hello.py
```

a poté skript přímo spustit:

```
./hello.py
```

Příkaz chmod a+x nastaví právo na spouštění pro všechny uživatele. O to, že je potom skript zpracován korektním interpretem, se stará první řádek skriptu. I když se zdá býti komentářem, tak se jedná o příkaz pro shell, který shellu říká, kde lze nalézt interpret pro korektní zpracování dalších řádků skriptu. Zobrazí-li se na obrazovce „Hello world!!!“, tak vše proběhlo v pořádku a Python je připraven k použití. V okamžiku, kdy je Python připraven, tak už schází pouze poslední dvě součásti embedded systému a těmi jsou HTTP server a obslužný program. Http server se instaluje opět podobným způsobem, jako předchozí balíčky a to příkazem:

```
opkg install mini-httpd*mipsel.ipk
```

Po samotné instalaci serveru je ještě server nutné nakonfigurovat a spustit. Konfigurační soubor se nachází v */etc/mini_httpd.conf* a lze jej ponechat v původním nastavení. Zapotřebí je ovšem vytvoření složky pro CGI. CGI je protokol který umí spolupracovat s vnějšími programy, které mohou generovat dynamický obsah stránky a tento je poté pomocí CGI zpracován na statický. Takový je i zobrazen uživateli ve webovém prohlížeči. Vytvoření potřebné složky je možné již známým příkazem:

mkdir /www/cgi-bin

Do této složky budou později přidány skripty jazyka Python obsahující kód obsahu stránek. Jestliže je již potřebná složka vytvořená, tak nic nebrání prvnímu spuštění web serveru. Tento webserver bude pro začátek vypisovat pouze chybný stav, jelikož mu zatím chybí kód stránek. Spuštění se provede stejným způsobem, jako v případě SSH serveru a to konkrétně:

/etc/init.d mini_httpd start

Stejným způsobem, jako v případě SSH je také zapotřebí vytvořit symbolický link na tento soubor v rc.d, aby se server spouštěl automaticky po startu. Tedy přejít do složky rc.d:

cd /etc/rc.d

a spustit příkaz vytváření odkazu:

```
ln –s ../init.d/mini_httpd S50mini_httpd
```

Tímto je ukončena instalace web serveru a lze již přejít k jednotlivým programům či skriptům, jež se starají o chod celého systému. Jako nejzákladnější a nejdůležitější program je program sběru dat z rozhraní GPIO. Tento je inspirován programem

gpioctl, zmíněným výše při kompilaci systému. Celý program je poměrně jednoduchý a jako jediný je napsán v jazyce C. Toto rozhodnutí bylo učiněno ze dvou důvodů. Prvním z nich je rychlost kódu jazyka C, která je stále ještě větší než jaké může dosáhnout kterýkoliv skriptovací jazyk. Druhým důvodem potom byla složitost s jakou by musel být program v Pythonu tvořen. Většinou je rychlost a jednoduchost programování právě devizou skriptovacích jazyků, ale ne v případě nízkoúrovňového programování blízkého jádru. Celý níže uvedený program by mohl být také napsán v jazyce Python, ale jazyku Python nejsou poskytnuty konstanty, které se v jazyce C nacházejí. Jedná se o speciální konstanty pro zařízení, kterým jsou GPIO linky. Tyto konstanty jsou dostupné pouze na konkrétním systému a do jazyka Python by musely být přeneseny, což značně komplikuje situaci. Jazyk Python obecně podporuje spolupráci s jazykem C a v C si lze také programovat přídavný kód pro Python. Toto ovšem vyžaduje přítomnost kompileru, který na uvedeném systému není a nelze jej ani doinstalovat. Dle autorů OpenWrt se to ani nedoporučuje, jelikož každý vygenerovaný systém OpenWrt je svým způsobem unikátní a nelze vytvořit obecný kompiler, který by fungoval na většině zařízeních bez potíží. Jediný přítomný kompiler je ten v toolchain. Tomuto ovšem chybí přítomnost Pythonu a jeho zdrojových kódů. Bylo by jej sice možné doinstalovat do toolchain, ale tento způsob by byl zbytečně složitý a zdlouhavý. Mnohem jednodušším způsobem je využití možnosti jazyka Python zpracovávat sdílené (shared) a DLL knihovny. Tento způsob byl uveden jako jeden z mnoha v [19], ale byl nejjednodušší a v dané situaci také nejlépe proveditelný. Autor tohoto textu jako jedinou nevýhodu uvedeného způsobu uvádí nutnost nastavení jedné ze systémových proměnných (LD_LIBRARY_PATH) na cestu k dané knihovně. Tato nevýhoda byla ovšem eliminována načítáním knihovny absolutní cestou, nikoliv pouze jejím jménem.

Samotný program pod názvem *gpiolib.c*, který bude později zpracován a použit jako knihovna je následující:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <linux/gpio_dev.h>
#include <linux/ioctl.h>

char inport(int pin){
int fd;
if ((fd = open("/dev/gpio", O_RDWR)) < 0){
       return -1;
       }
ioctl(fd, GPIO_DIR_IN, pin);
close (fd);
return 0;
}
char outport(int pin){
int fd;
if ((fd = open("/dev/gpio", O_RDWR)) < 0){
       return -1;
       }
ioctl(fd, GPIO_DIR_OUT, pin);
close (fd);
return 0;
}
char get(int pin){
int fd;
char result;
if ((fd = open("/dev/gpio", O_RDWR)) < 0){
       return -1;
       }
result = ioctl(fd, GPIO_GET, pin);
close (fd);
return result;
}
char set(int pin){
int fd;
if ((fd = open("/dev/gpio", O_RDWR)) < 0){
       return -1;
       }
ioctl(fd, GPIO_SET, pin);
close (fd);
return 0;
}
char clear(int pin){
```

```
int fd;
if ((fd = open("/dev/gpio", O_RDWR)) < 0){
	return -1;
	}
ioctl(fd, GPIO_CLEAR, pin);
close (fd);
return 0;
}
```

Jak je vidět jedná se o poměrně jednoduchou knihovnu obsahující pouze pět funkcí a to pro nastavení GPIO pinů, jako vstupu nebo výstupu a dále potom zápis a čtení jejich úrovní. Základem celé knihovny je zařízení */dev/gpio*, které bylo vytvořeno v systému aplikací jednoho z patchů během kompilace. Základem celé práce s tímto zařízením je jeho obsluha knihovní funkcí ioctl, kterou jsou zapisovány na piny GPIO požadované hodnoty nastavení – zde uvedené jako konstanty. Tyto konstanty jak již bylo řečeno zabránili implementaci této části kódu v jazyce Python.

Aby bylo možno tuto knihovnu používat, je třeba ji nejdříve jako knihovnu zkompilovat a to pomocí kompileru GCC, který jak již bylo řečeno není součástí cílového systému a nachází se pouze v toolchain. Na hostujícím systému je tedy zapotřebí přejít do složky s toolchain:

```
cd /home/openwrt/data/staging_dir/toolchain-mipsel_gcc*/usr/bin
```

a pomocí GCC knihovnu zkompilovat následujícím příkazem:

```
mipsel-openwrt-linux-gcc –o gpiolib.so –shared –fPIC gpiolib.c
```

Příkaz pro tvorbu sdílené knihovny je použit přímo z [19]. Takto vytvořenou knihovnu je zapotřebí přesunout na cílový systém. Nejlepším způsobem je využití SSH serveru a to konkrétně funkce scp:

```
scp gpiolib.so root@192.168.1.100:/www/cgi-bin
```

Syntaxe tohoto příkazu je poměrně jednoduchá a na první pohled zřejmá. Nicméně, root je uživatel pod kterým se na SSH server připojuje, 192.168.1.100 je IP adresa SSH serveru a */www/cgi-bin* je cesta k adresáři do kterého má být daný soubor zkopírován. Stejným způsobem lze tento příkaz použit i pro další skripty.

Za účelem využití výše vytvořené knihovny v programech jazyka Python byl vytvořen skript *gpioctl.py*:

```
#!/usr/bin/python

from ctypes import cdll
```

```
from ctypes import c_char
import os

gpiolib = cdll.LoadLibrary(os.path.join(os.getcwd(),"gpiolib.so"))

def gpioin(pinNumb):
	if (gpiolib.inport(pinNumb))
		print ("GPIO device can't be open.")
		return 1
	else:
		return 0

def gpioout(pinNumb):
	if (gpiolib.outport(pinNumb)):
		print ("GPIO device can't be open.")
		return 1
	else:
		return 0

def gpioset(pinNumb):
	if (gpiolib.set(pinNumb)):
		print ("GPIO device can't be open.")
		return 1
	else:
		return 0

def gpioclear(pinNumb):
	if (gpiolib.clear(pinNumb)):
		print ("GPIO device can't be open.")
		return 1
	else:
		return 0

def gpioget(pinNumb):
	ret = gpiolib.get(pinNumb)
	if ret == -1:
		print ("GPIO device can't be open.")
		return 1
	else:
		return ret
```

I v případě tohoto skriptu se jedná o jednoduchý program, který pouze kontroluje hodnoty získané z knihovny a v případě volání je předává do dalších skriptů. Prvním důležitějším skriptem je skript *dbhandler.py*:

```
#!/usr/bin/python

from sqlite3 import dbapi2 as sqlite
from time import strftime
import os

def firstTime():
    try:
        db = sqlite.connect ("data")
        db.execute("create table ad(time time, value integer)")
    except:
        print("Some database problem during it's creation!")
        return 1
    else:
        db.commit()
        db.close()
    return 0

def add(value):
    time = strftime("%Y-%m-%d %H:%M:%S")
    try:
        db = sqlite.connect ("data")
       db.execute("insert   into   ad(time,   value)   values(:time,
      :value)",{"time" : time,"value" : value})
    except:
       print  ("Some  problem  in  table  ad  during  insertion  of  new
      element!")
        return 1
    else:
        db.commit()
        db.close()
        return 0

def remove(timecond = "= *"):
    try:
        db = sqlite.connect ("data")
        db.execute("delete from ad where time %s"%timecond)
        changes = db.total_changes
```

```
        except:
            print ("Some problem in table ad during element removing!")
            return 1
        else:
            db.commit()
            db.close()
            return changes

    def read(timecond = "*"):
        sample = []
        try:
            db = sqlite.connect ("data")
            out = db.execute("select %s from ad"%timecond)
            for timedb, valuedb in out:
                sample.append(timedb)
                sample.append(valuedb)
        except:
            print ("Some problem in table ad during it's reading!")
            return 1
        else:
            db.commit()
            db.close()
        return sample

    if not os.path.isfile(os.path.join(os.getcwd(),"data")):
        firstTime()
```

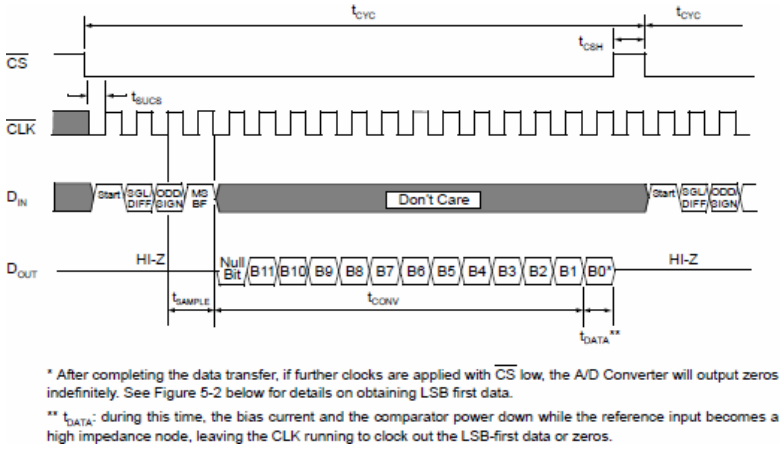
Tento skript už je mírně složitější a stará se o práci s databází. Obsahuje funkci, která automaticky vytváří databázi, pokud ještě neexistuje – firstTime() , dále funkci pro přidávání – add(), čtení – read() a mazání dat z této databáze – remove(). Všechny funkce vracejí jako svou návratovou hodnotu chybový stav nebo neutrální hodnotu – 0 a nebo v případě funkce remove() počet smazaných záznamů a v případě read() všechny přečtené záznamy. Nyní lze data načítat i ukládat, zbývá tedy pouze chybějící mezičlánek starající se o komunikaci a zpracování dat z A/D převodníku. Tímto mezičlánkem je skript *adhandler.py* obsahující následující kód:

```
#!/usr/bin/python
import gpioctl,time

def clock():
    gpioctl.gpioout(18) #CLK signal
```

```
    gpioctl.gpioclear(18)
    time.sleep(0.001)
    gpioctl.gpioset(18)

    return 0

def chipselect(start = 1):
    gpioctl.gpioout(21) #CS signal
    if start:
        gpioctl.gpioclear(21)
    else:
        gpioctl.gpioset(21)
    time.sleep(0.001)
    gpioctl.gpioset(21)
    return 0

def config():
    gpioctl.gpioout(12) #DIN signal
    chipselect()
    clock()
    gpioctl.gpioclear(12) #Start bit
    clock()
    clock() #Single ended mode - no change
    gpioctl.gpioset(12)   #Channel 0
    clock()
    clock() #MSBF to 0 - only from MSB to LSB order (no change)
    return 0

def data():
    bitword = []
    gpioctl.gpioin(15)
    clock() #ignore null bit
    for i in range(11):
        bitword.append(gpioctl.gpioget(15))
        clock()

    resolution = 4096
    result = 0
    for i in range(11):
        result = result + bitword[i]*resolution
        resolution = resolution / 2
    return result
```

```
def main():
    clock()
    config()
    result = data()
    chipselect(0)
    return result

main()
```

Celý výše uvedený skript se stará pouze o implementaci časování z následujícího obrázku:



* After completing the data transfer, if further clocks are applied with $\overline{CS}$ low, the A/D Converter will output zeros indefinitely. See Figure 5-2 below for details on obtaining LSB first data.

** t_{DATA}: during this time, the bias current and the comparator power down while the reference input becomes a high impedance node, leaving the CLK running to clock out the LSB-first data or zeros.

Obrázek 8 Časování komunikace s obvodem MCP3202 [14]

Tento skript je rozdělen do několika malých podfunkcí, z nichž clock() se stará o generování hodinového signálu, chipselect() generuje signál CS. Config() potom generuje, s pomocí předešlých funkcí, data pro nakonfigurování A/D převodníku na kanál nula v klasickém režimu a bez opakování výstupu. Předposlední funkcí je data(), která se stará o načtení dat z převodníku. Nakonec funkce main() zastřešuje všechny předešlé funkce a volá je v korektním pořadí. Jako návratovou hodnotu vrací skript hodnotu získanou z A/D převodníku, která může být dále zpracována jiným skriptem – například přepočet na napětí, teplotu a další. Celý přepočet získaných binárních dat do desítkové podoby je jednoduchý. Každý bit se násobí svou hodnotou své pozice (11. bit hodnotou 2^{11} atd.) a dílčí výsledky jsou poté sečteny. Tímto je vytvořeno jádro celého systému. Nyní lze přejít k webovému serveru a kódu generujícímu obsah webových stránek.

První webová stránka bude pouze se statickým kódem html a bude sloužit k přesměrování uživatele na stránky s dynamickým obsahem. Tato statická stránka je automaticky hledána serverem v okamžiku žádosti o připojení, proto je její tvorba důležitá. Její kód je jednoduchý a vypadá následovně:

```
<html><head>
	<meta content = “text{html; charset=ISO-8859-1”>
	<meta http-equiv “REFRESH” content= “0; url=file:./cgi-bin/main.py”>
	<title>index</title>
</head><body>
Redirecting...
</body></html>
```

Uvedená stránka se odkazuje na skript *main.py*. Tento skript se již stará o skutečnou domovskou stránku webového serveru a jeho kód vypadá následovně:

```
	#!/usr/bin/python

	print "Content-type: text/html"
	print

	import cgi, adhandler

	print "<html>"
	print "<head>"
	print "<title>Embedded System For Acquiring Data</title>"
	print "</HEAD>"
	print "<body bgcolor = white>"
	print "<div style='text-align: center;'>"
	print "<h2>Embedded System For Acquiring Data</h2>"
	print "<img style='width: 300px; height: 182px;' alt='Edimax BR6104KP
	inside' src='../edimax.jpg'>"
	print "<br><br>"
	print "<div style='text-align: left;'>"
	print "Actual value on A/D converter's outputs: "
	value = adhandler.main()
	print "<h4>", value, "</h4>"
	print "<br><br>"
	print "<a href ='database.py'>Database Data</a >"
	print "<br><br><br><br><br><br><br><br>"
	print "<div style='text-align: center;'>"
	print "Powered by:"
	print "<br>"
```

```
print "<img style='width: 211px; height: 71px;' alt='python'
src='../python.gif'><br>"
print "</body>"
print "</html>"
```

Víceméně se v tomto skriptu jedná o tisk html kódu stránky, pouze s výjimkou jednoho řádku, kde jsou načtená aktuální data z A/D převodníku a zobrazena. Uživatel tak každým obnovením stránky získá čerstvá data. Výsledek je uveden na Obrázek 9. Stránka také obsahuje odkaz na druhou, již zajímavější stránku.

Obrázek 9 Hlavní stránka

Tato stránka zobrazuje data přímo z databáze, dovoluje ukládat data nová a také mazat již uložená pomocí volitelného filtru. Kód této stránky je uveden ve skriptu *database.py* a vypadá následovně:

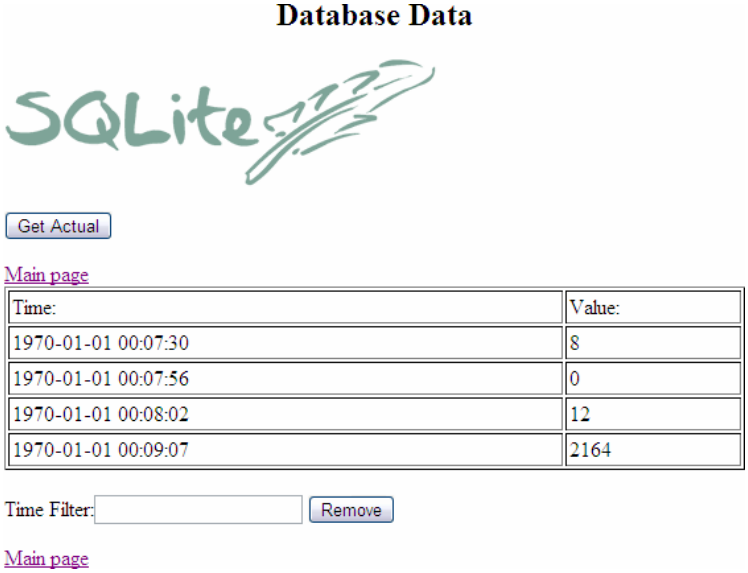
```
#!/usr/bin/python

import cgi,dbhandler,adhandler

def page(changes = 0):
    print "<html>"
    print "<head>"
    print "<title> Database data </title>"
    print "</HEAD>"
    print "<body bgcolor = white>"
    print "<div style='text-align: center;'>"
    print "<h2>Database Data</h2>"
    print "<div style='text-align: left;'>"
```

```
    print "<img style='width: 327px; height: 97px;' alt='SQLite'
src='../SQLite.gif'>"
    print "<br><br>"
    print "<form method=post action='database.py'>"
    print "<input type=hidden name='update' value='display'>"
    print "<input type=submit value='Get Actual'>"
    print "</form>"
    print "<a href ='main.py'>Main page</a >"
    if not changes == 0:
        print "%s items removed from database"
    print "<table style='text-align: left; width: 100%;' border='1'
cellpadding='2' cellspacing='2'>"
    print "<tbody>"
    print "<tr>"
    print "<td>Time:</td>"
    print "<td>Value:</td>"

    data = dbhandler.read()
    for i in range(len(data)):
        if i%2 == 0:
            print "<tr><td>"
            print data[i]
            print "</td>"
        else:
            print "<td>"
            print data[i]
            print "</td></tr>"

    print "</tbody>"
    print "</table>"
    print "<br>"
    print "<form method=post action='database.py'>"
    print "Time Filter:<input type=text name='time'>"
    print "<input type=hidden name='remove' value='display'>"
    print "<input type=submit value='Remove'>"
    print "</form>"
    print "<a href ='main.py'>Main page</a >"
    print "<br><br><br><br><br>"
    print "Note: Time condition can be any of SQL commands for time
variable."
    print "For example: < '2010-12-21 12:21:12' OR '< 'now' and etc."
    print "Apostrophes are required as a part of filter!"
```

```
        print "With empty filter all data will be deleted!"
        print "</body>"
        print "</html>"

    def main():
        print "Content-type: text/html"
        print

        form = cgi.FieldStorage()

        if (form.has_key("update")):
            dbhandler.add(adhandler.main())
            page()
        elif(form.has_key("remove")):
            dbhandler.remove(form["time"].value)
            page()
        else:
            page()

    main()
```

V uvedeném kódu se nacházejí dva formuláře s tlačítky. Kliknutí na kterékoliv z těchto tlačítek vyvolá znovu načtení této stránky s tím rozdílem, že budou načtena i data určující, na které tlačítko bylo kliknuto. Na tuto akci je vyvolána reakce v podobě načtení nové hodnoty do databáze nebo vymazání položek z databáze splňujících podmínku ve filtru. Výsledná stránka v ostrém provozu je uvedena na Obrázek 10. Aby dva výše uvedené skripty pracovali korektně, resp. je mohl protokol CGI spouštět je třeba jim dát práva pro spuštění. V opačném případě se v prohlížeči při pokusu o načtení stránky zobrazí zpráva o chybě. Jelikož se ovšem změna práv pro spouštění hodí obecně na kterýkoliv z výše uvedených skriptů, tak je vhodné tuto změnu provést globálně pro celou složku následujícím způsobem:

```
chmod a+x /www/cgi-bin/*.py
```



Obrázek 10 Stránka s databází

Tímto je ukončena celá kapitola o zprovozňování systému. Po uvedené konfiguraci systém pracoval zcela bez potíží i s menší rezervou paměti RAM. Při maximálním vytížení se podařilo hodnotu paměti RAM snížit až na pouhých 600kB volné paměti. Toto číslo je mírně znepokojující a v případě jakéhokoliv dalšího rozšíření by bylo vhodné USB disk rozdělit na dva oddíly (například pomocí programu fdisk) a jeden z těchto oddílů vyhradit odkládacímu prostoru – Swap. Obecnou zásadou je používat oddíl swap o 1,5 násobku velikosti paměti RAM. Podpora v jádře pro oddíl tohoto typu nebyla odstraněna a tak stačí pouze změnit nastavení konfiguračního souboru */etc/fstab* tak, aby obsahoval nový diskový oddíl se systémem souborů typu swap.

6. ZÁVĚR

Uvedená diplomová práce se komplexně zabývá návrhem a tvorbou embedded systémů, konkrétně potom embedded systémů pro sběr dat. I když je uveden pouze jeden ucelený postup tvorby cílového embedded systému, tak je v každé části práce uvedeno množství dalších možných směrů, kterými je možno se vydat. Každá část práce se také nesnaží býti orientována pouze na konkrétní embedded systém, ale popisuje veškeré činnosti takovým způsobem, aby mohly být aplikovány na jakýkoliv obecný embedded systém. Výsledkem je tedy avizovaný komplexní popis návrhu a tvorby, ze kterého může vzniknout několik odlišných embedded systémů pro sběr dat a také zcela nových embedded systémů.

Samotný vytvořený systém je díky aplikaci operačního systému GNU/Linux a také poměrně velkého počtu vstupů a výstupu velice modulární. Pouhým přidáním dalšího skriptu napsaného v jazyce Python a případným rozšířením vstupů/výstupů pomocí sério-paralelních a paralelně-sériových registrů lze získat systém, který může být určen k některým automatizovaným činnostem. Příkladem budiž řízení klimatizace. Dále může sloužit jako zabezpečovací ústředna domu, která by se starala o automatické zapínání světel v domě, za nepřítomnosti majitele a také k jeho informování například pomocí připojeného telefonu formou SMS, přes síť Internet a nebo e-mailem. Dalším využitím by mohlo být vzdálené ovládání zařízení v domácnosti, jako například zapnutí topení před odchodem z práce. Samotný sběr dat může být také zobrazován na displeji připojeném přímo k dalším volným výstupům. Zařízení podobná uvedenému slouží i statikům pro měření tlaků v domě a jejich monitorování na dálku pomocí sítě Internet. Možnosti uvedeného zařízení jsou tedy opravdu široké a za uvedené náklady lze jen těžko sehnat v komerční sféře něco podobného, zda-li vůbec. Zřejmě jediným možným vylepšením by bylo rozšíření množství vstupů a výstupů pomocí registrů a zobecněním obslužného programu tak, aby mu bylo možno pouze předávat jako vstupní parametry čísla vstupů/výstupů a operace (funkce) jejich zpracování. Lze si také představit pomocné vývojové prostředí podobné například LabView, jež by obsahovalo bloky s operacemi a bloky vstupů a výstupů. Spojením těchto bloků by poté byl vygenerován program, jež by se

nahrál pomocí SSH protokolu do embedded zařízení a funkce zařízení by mohla být změněna ke zcela jinému účelu. Ze všech uvedených možností nasazení a možností rozšíření je jasně patrná výhoda použití operačního systému, jež vývoji embedded systémů dodává rychlost a jednoduchost. I proto není divu, že o nasazení operačních systémů a speciálně systémů GNU/Linux se v současné době zajímají ty největší společnosti na komerčním trhu, jako například internetový gigant Google.

Celé zařízení ihned po nastavení pracovalo zcela korektně dle zadání. Data byla sbírána pomocí uvedeného A/D převodníku a ukládána do databáze. Tato data je možno zobrazit na kterémkoliv PC disponujícím připojením do stejné sítě nebo v případě sítě Internet kdekoliv na světě. Konfigurace, kalibrace, ale také změny funkcionality mohou být provedeny pomocí vzdáleného spojení protokolem SSH. Výsledný embedded systém a práce samotná ilustrují obrovskou sílu a možnosti aplikací těchto systémů nejen v komerční sféře, ale také v průmyslu.

7. LITERATURA

[1] JELÍNEK, Lukáš. *Vytváříme vlastní distribuci Linuxu*. vydání první. Brno : Computer Press, a.s., 2010. 304 s. ISBN 978-80-251-2433-8.

[2] ŠIMŮNEK, Milan. *SQL : Kompletní kapesní průvodce*. Vydání 1. Praha : Grada Publishing, spol. s r.o., 1999. 248 s. ISBN 80-7169-692-7.

[3] KRČMÁŘ, Petr. *Linux : postavte si vlastní počítačovou síť*. První vydání. Praha : Grada Publishing, a.s., 2008. 184 s. ISBN 978-80-247-1290-1.

[4] SATRAPA, Pavel. *Perl pro zelenáče*. II. vydání. Praha : Neocortex, s. r. o., 2001. 224 s. ISBN 80-86330-02-8.

[5] HARMS, Daryl; MCDONALD, Kenneth. *Začínáme programovat v jazyce Python*. 2. opravené vydání. Brno : Computer Press, a.s., 2008. 456 s. ISBN 978-80-251-2161-0.

[6] O'KEEFE, Greg. *ANU College of Engineering & Computer Science* [online]. V0.9a. [Austrálie] : November 2000, November 2000 [cit. 2010-05-01]. *From Power Up To Bash Prompt.* Dostupné z WWW: <http://users.cecs.anu.edu.au/~okeefe/p2b/power2bash/power2bash.html>.

[7] O'KEEFE, Greg. *ANU College of Engineering & Computer Science* [online]. V0.8. [Austrálie] : September 2000, September 2000 [cit. 2010-05-01]. *How To Build a Minimal Linux System from Source Code.* Dostupné z WWW: <http://users.cecs.anu.edu.au/~okeefe/p2b/buildMin/buildMin.html>.

[8] WILMS, Tom. *Studenten Homepages [online]*. Nizozemsko : 2005-08-09, 2005-08-09 [cit. 2010-05-01]. *Adm5120 toolchain, kernel & root fs HOWTO.* Dostupné z WWW: <http://www.student.tue.nl/Q/t.f.a.wilms/adm5120/>.

[9] OLIVER, Ryan, et al. *Cross-Compiled Linux From Scratch - Embedded : MIPS* [online]. Version SVN-0.0.1-20090726-mips. [s.l.] :

[s.n.], 26.07.2009, 26.07.2009 [cit. 2010-05-01]. Dostupné z WWW: <http://cross-lfs.org/view/clfs-embedded/mips/>.

[10] KORSGAARD, Peter, et al. *Buildroot: making Embedded Linux easy* [online]. 1999, 26 February 2010 [cit. 2010-05-01]. Dostupné z WWW: <http://buildroot.net/>.

[11] *OpenWrt* [online]. 1995, 2010 [cit. 2010-05-01]. Dostupné z WWW: <http://www.openwrt.org/>.

[12] *Kelvin's Thunderstorm Embedded geekery and similar pursuits* [online]. September 30, 2009, September 30, 2009 [cit. 2010-05-01]. *Omnima Embedded Controller and OpenWRT*. Dostupné z WWW: <http://www.kelvinsthunderstorm.com/omnima-embedded-controller-and-openwrt/>.

[13] *Forum OpenWrt* [online]. 2009-01-23, 2010-03-05 [cit. 2010-05-01]. *USB-rootfs on boot on Edimax BR-6104KP (ADM5120) [WORKS]*. Dostupné z WWW: <https://forum.openwrt.org/viewtopic.php?id=18570&p=1>.

[14] *MCP3202 : 2.7V Dual Channel 12-Bit A/D Converter with SPI Serial Interface* [online]. Revision D (December 2006). USA : Microchip Technology Inc., 10/19/06, 10/19/06 [cit. 2010-05-20]. Dostupné z WWW: <http://ww1.microchip.com/downloads/en/DeviceDoc/21034D.pdf>.

[15] MACHO, Tomáš. *Mikroprocesory*. Brno : FEKT Vysokého učení technického v Brně, 20. 5. 2009. 113 s.

[16] MARSHALL, Graham. *Sunspot* [online]. Feb 2006, 20 March 09 [cit. 2010-05-20]. Sweex router project. Dostupné z WWW: <http://www.sunspot.co.uk/Projects/sweexproject.htm>.

[17] *Linux-mips* [online]. 30 December 2009, 30 December 2009 [cit. 2010-05-20]. Adm5120 - LinuxMIPS. Dostupné z WWW: <http://www.linux-mips.org/wiki/Main_Page>.

[18] BEEKMANS, Gerard , et al. *Linux from scratch* [online]. 1998, 2010/03/01 [cit. 2010-05-20]. Linux from scratch. Dostupné z WWW: <http://www.linuxfromscratch.org >.

[19] MERTZ, David. Wrapping C/C++ for Python [online]. [s.l.] : [s.n.], [2008] [cit. 2010-05-20]. Dostupné z WWW: <http://teckla.idyll.org/~t/transfer/public/day3.pdf>.

[20] KAINKA, Burkhard; BERNDT, Hans-Joachim. Využití rozhraní PC pod Windows. 1. české vydání 2000. Příbram : HEL, 2000. 152 s. ISBN 80-86167-13-5.

SEZNAM ZKRATEK

Zkratka/Symbol	Popis
USB	Universal Serial Bus – Univerzální sériová sběrnice
GPIO	General Purpose Input/Output – Vstupy/Výstupy pro obecné použití
OS	Operating System - Operační systém
PDA	Personal Digital Assistant – Osobní digitální asistent
FBW	Fly By Wire
ABS	Anti-Lock Brake System – Systém proti zablokování brzd
CPU	Central Processor Unit – Centrální procesorová jednotka
RAM	Random Access Memory – Paměť s náhodným přístupem
HDD	Hard Disk Drive – Disková jednotka
ROM	Read Only Memory – Paměť pouze pro čtení
DSP	Digital Signal Processor – Digitální signálový procesor
FFT	Fast Fourier Transform – Rychlá Fourierova transformace
HW	Hardware
SW	Software
PC	Personal Computer – Osobní počítač
A/D	Analog To Digital – Analogově/číslicový
CGI	Common Gateway Interface
SPI	Serial Peripheral Interface – Sériové periférní rozhraní
PID	Process Identifier – Identifikační číslo procesu
IMEI	International Mobile Equipment Identity – Mezinárodní identifikační číslo mobilního zařízení

DVD	Digital Video Disc – Digitální video disk
LED	Light Emitting Diode – Světlo vyzařující dioda

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

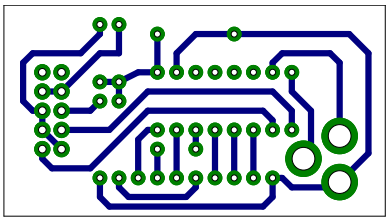
Příloha 3 Návrh desky plošných spojů vstupně/výstupní desky

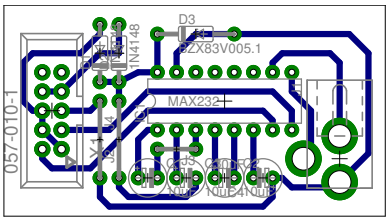
Příloha 4 Osazovací plánek vstupně výstupný desky

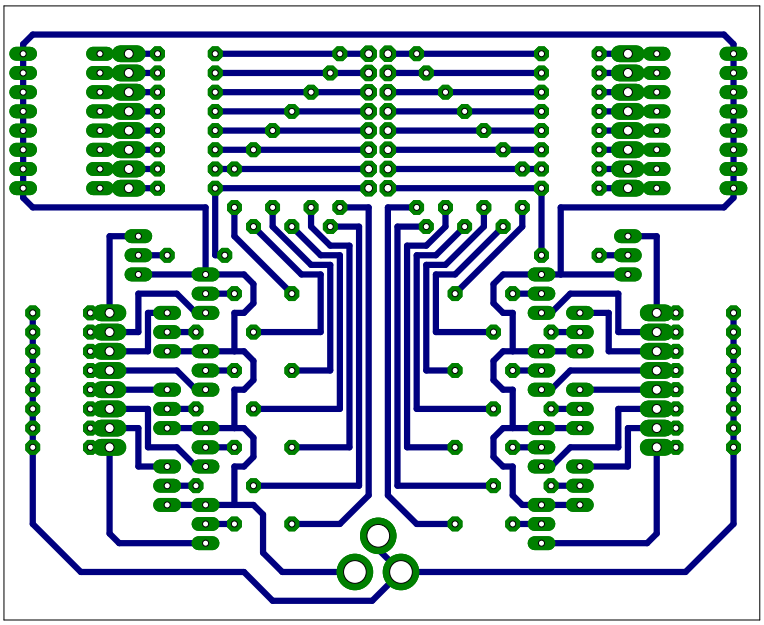
Příloha 5 Návrh desky plošných spojů A/D převodníku

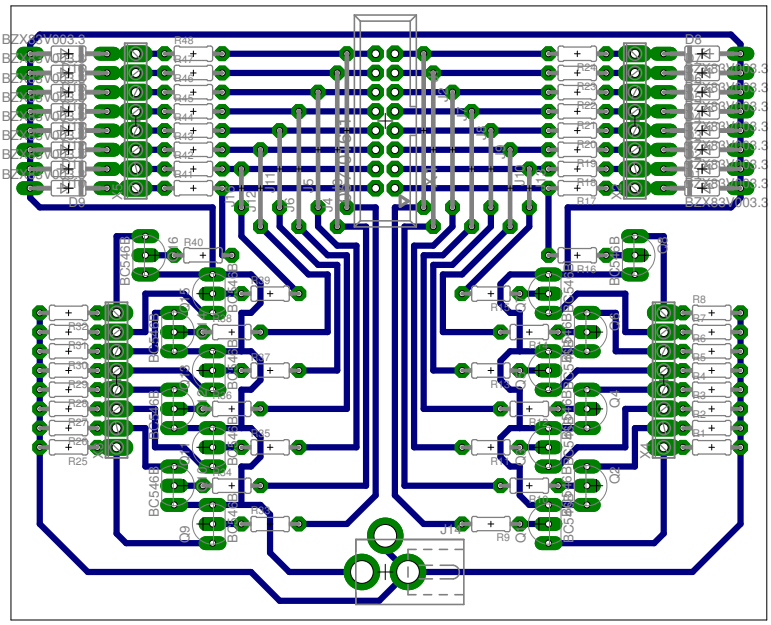
Příloha 6 Osazovací plánek desky A/D převodníku

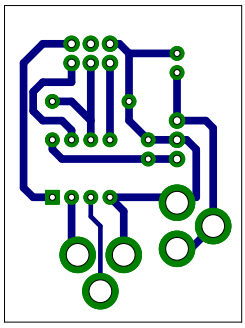
Příloha 7 Seznam součástek

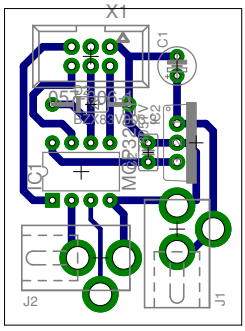



D3
BZX83V005.1
1N4148
MAX232
057-0110-1








X1
C1
C1
J2
J1

Seznam Součástek:

Převodník PC/Router:

IC1	MAX232
D1,D2	1N4148
D3	BZX83V005.1
C1-C4	10uF/25V
J1	JACK 3.5mm
X1	MLW10G

Vstupně/Výstupní deska:

D1-D16	BZX83V003.3
T1-T16	BC546B
R1-R8	470R
R9-R24	1k
R25-R32	470R
J1	JACK 3.5mm
X1	MLW16G
X2-X5	ASS10520G

A/D Převodník:

IC1	MCP3202
IC2	7805
D1	BZX83V003.3
C1	10uF/25V
C2	100nF/50V
J1,J2	JACK 3.5mm
X1	MLW6G