VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

# DIPLOMOVÁ PRÁCE

Brno, 2016 Bc. Luboš Rácek


# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

## FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

## ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS


# MOŽNOSTI ZABEZPEČENÍ MEZI-PROCESOROVÉ KOMUNIKACE PRO DEDIKOVANÁ ZAŘÍZENÍ
POSSIBILITIES OF SECURING INTER-PROCESSOR COMMUNICATION FOR DEDICATED DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS


AUTOR PRÁCE
AUTHOR
Bc. Luboš Rácek

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. David Troják


BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

# Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

***Student:*** Bc. Luboš Rácek ***ID:*** 148148
***Ročník:*** 2 ***Akademický rok:*** 2015/16

**NÁZEV TÉMATU:**

## Možnosti zabezpečení mezi-procesorové komunikace pro dedikovaná zařízení

**POKYNY PRO VYPRACOVÁNÍ:**

Seznamte se s SoC rodiny i.MX a s možnostmi mezi-procesorové komunikace. Vyberte vhodnou platformu např. i.MX6Q, 6SX nebo 7D SDB podpořenou build systémem Yocto. Zvažte možnosti zabezpečení mezi-procesorové komunikace na této platformě. Navrhněte, realizujte a zdokumentujte ukázkovou aplikaci. Výsledkem práce bude funkční demonstrační aplikace se zabezpečeným přenosem mezi procesory.

**DOPORUČENÁ LITERATURA:**

[1] I.MX SoC family. I.MX Applications Processors based on ARM® Cores [online]. Austin, TX: NXP Semiconductors, 2016 [cit. 2016-03-16]. Dostupné z: http://www.nxp.com/iMX

[2] Linux, Android and RPMsg Documentation. I.MX 6 Series Software and Development Tool|NXP [online]. Austin, Texas: NXP Semiconductors, 2016. Dostupné z: http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx6qp/i.mx-6-series-software-and-development-tool-resources:IMX6_SW#nogo

***Termín zadání:*** 1.2.2016 ***Termín odevzdání:*** 25.5.2016

***Vedoucí práce:*** Ing. David Troják
***Konzultant diplomové práce:*** Ing. Jiří Kotzian, Ph.D.

**doc. Ing. Jiří Mišurec, CSc.**, *předseda oborové rady*

## ABSTRAKT

Diplomová práce se zaměřuje na uvedení do problematiky mezi-procesorové komunikace. Věnuje se Linuxové distribuci vytvořené projektem Yocto na vybrané platformě z řady mikrokontrolérů i.MX společnosti NXP a řešení zabezpečení mezi-procesorové komunikace.

## KLÍČOVÁ SLOVA

Operační systém, Linux, dedikovaná zařízení, YOCTO, Linuxová distribuce, mezi-procesorová komunikace, zabezpečení

## ABSTRACT

Diploma thesis focuses on introduce problematic issue of inter-core communication, implementation of Linux distribution made by project Yocto on chosen platform of NXP i.MX microcontrolers family and solution for secured inter-core communication.

## KEYWORDS

Operating system, Linux, dedicated devices, YOCTO, Linux distribution, inter-core communication, security

RÁCEK, Luboš *Možnosti zabezpečení mezi-procesorové komunikace pro dedikovaná zařízení*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015. 50 s. Vedoucí práce byl Ing. David Troják

# PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Možnosti zabezpečení mezi-procesorové komunikace pro dedikovaná zařízení“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské 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(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ..... .....

podpis autora(-ky)

# PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu semestrální práce panu Ing. Davidu Trojákovi a panu Ing. Jiřímu Kotzianovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.


Brno ..... .....

podpis autora(-ky)

# OBSAH

# SEZNAM OBRÁZKŮ

# ÚVOD

Cílem diplomové práce je seznámení s jednočipovými systémy rodiny i.MX společnosti NXP, možnostmi mezi-procesorové komunikace a její zabezpečení. Dále seznámení s problematikou nasazení operačního systému Linux na dedikovaná zařízení a následná implementace vlastní distribuce operačního systému vytvořeného pomocí projektu Yocto.

Pro správné uvedení do tématu meziprocesorové komunikace je nutné se nejdříve věnovat teorii operačních systémů, kde se meziprocesorová komunikace využívá. Pro vhled do problematiky se teoretická část věnuje historii operačních systémů, jejich rozdělením a problematice operačních systémů pro dedikovaná zařízení. Není zde ani opomenuta teorie mezi-procesorové komunikace.

V práci bude představen jednočipový systém i.MX 6SoloX společnosti NXP, osazeném na vývojové desce Sabre SDB. Meziprocesorová komunikace bude probíhat mezi jádry tohoto mikrokontroléru, a to Cortex M4 a Cortex A9. Pro jádro Cortex A9 bude sestaven operační systém Linux pomocí projektu Yocto, na kterém bude spuštěna demonstrační aplikace prezentující použití zabezpečené komunikace.

Uvedeme si možnosti zabezpečení, které vybraný mikrokontrolér nabízí a navrhneme možná řešení zabezpečení mezijádrové komunikace. Pro jádro Cortex M4 bude vytvořena aplikace postavena na systému FreeRTOS. Protokol pro meziprocesorovou komunikaci bude zvolen RPMSG. Zabezpečení protokolu bude provedeno pomocí softwarové knihovny implementující algoritmus Diffie Hellman, kterým se získá klíč pro symetrické šifrování. Tímto klíčem bude následná komunikace šifrována.

# 1 OPERAČNÍ SYSTÉMY

## 1.1 Historie operačních systémů

Operační systémy byly navrhovány jako sada pravidel pro programátory na míru použitého počítače. Sloužily především pro usnadnění práce dalších programátorů, protože ti se již nemuseli zabývat složitými nízko úrovňovými operacemi jako řízení hardwaru. Operační systém definuje veškeré funkcionality systému na základě hardwaru. Jestliže je chyba v operačním systému, selžou také aplikace využívající nesprávně nakonfigurovanou funkci operačního systému. Evoluce operačních systémů je velice obsáhlá, operační systémy prošly celou řadou změn a úprav většinou podle potřeb jeho uživatele. Vybrané operační systémy, které znamenaly jistý pokrok během vývoje jsou například tyto:[1]

**GM-NAA I/O**

Operační systém jež vyvinul Robert L. Patrick ve společnosti General Motors v roce 1956 pro jejich sálový počítač IBM 704. Hlavním účelem operačního systému bylo zajištění automatického přechodu na další úkol po skončení předešlého. Operační systém byl použit přibližně na 40 těchto sálových počítačích.

**MCP**

Neboli Master Control Program. Vyvinut společností Burroughs Corporations pro jejich sálový počítač B5000 v roce 1961. MCP je dodnes využíván v systémech Unisys ClearPath/MCP.[1]

**DOS/360**

Po letech striktně hardwarového byznysu se IBM pustilo do vývoje operačních systémů. V IBM se vyvinulo několik neúspěšných operačních systémů, až nakonec v roce 1966 vydali operační systém DOS/360 a jeho nástupce, jež společnost IBM posadilo na pozici řízení hardwarového i softwarového průmyslu.

**Unix**

Ken Thompson, Dennis Ritchie, Douglas McIlroy a Joe Ossanna, programátoři pracující pro společnost AT&T Bell Labs, vyvíjeli a později v roce 1969 vydali operační systém Unix. Zpočátku byl systém rozšířen v rámci společnosti AT&T a později také mezi školami a univerzitami. Operační systém je napsán v jazyce C, což umožňuje snadnější modifikace, nasazení i portaci.

**CP/M**

CP/M neboli Control Program/Monitor, později také Control Program for Microcomputers, vyvinul v roce 1973 Greg Kildall jako vedlejší projekt pro jeho společnost Digital Research. CP/M se stal populární v sedmdesátých letech a bylo pro něj vyvinuto mnoho aplikací včetně WordStar a dBase. Operační

systém byl portován na mnoho různých hardwarů. Ve skutečnosti také v IBM chtěli použít CP/M pro jejich nový osobní počítač, nakonec byl zvolen MS-DOS.[8]

**BSD**

Berkeley Software Distribution, operační systém vyvinuli v roce 1977 na Kalifornské univerzitě v Berkeley. Jedná se o variantu systému Unix založenou na dřívější verzi Unixu z Bellových laboratoří.

**MS-DOS**

Vyvinut společností Microsoft pro osobní počítače firmy IBM v roce 1981. Byl to první široce dostupný operační systém pro domácí uživatele. V roce 1985 Microsoft vydal operační systém Microsoft Windows, který jej proslavil ještě víc. Microsoft Windows nabídl uživateli grafické rozhraní, jež jej markantně rozšířilo.

**SunOS**

Sun Microsystems vyvinulo SunOS v roce 1982. Byla to velice populární varianta Unixu založena na systému BSD.

**Mac OS**

Vyvinut společností Apple Computer, Inc pro jejich nový produkt Macintosh home PC v roce 1984. Macintosh byl široce inzerován. Mac OS byl první operační systém s vestavěným GUI. To vedlo k vytvoření velice stabilního operačního systému a díky snadnému použití se dočkal přijetí širokou veřejností.

**OS/2**

Operační systém vyvinut za spolupráce společností IBM a Microsoft v roce 1987. Ačkoliv byl intenzivně propagován, velké popularity se nedočkal.

**Linux**

V roce 1991 vyvinul Linus Torvalds jako volnou variantu Unixu. Linux je dnes velmi široce vyvíjený Open Source projekt, který hraje velmi významnou roli v dnešním průmyslu serverů.

**Sun Solaris**

Sun Microsystems vyvinul tento operační systém částečně z předchozí verze SunOS v roce 1992.

**Windows**

Operační systémy Windows společnosti Microsoft začaly svou éru již v roce 1985 svou první verzí systému, kde uživatelské rozhraní bylo rozděleno do okénkové podoby. V roce 1995 byla vydána další verze, Windows95, která měla grafické uživatelské rozhraní již integrováno. V této verzi byla poprvé použita nabídka Start, což je dodnes dominantním prvkem systému. Dalšími nástupci byli Windows98 (1998), Windows 2000 (2000), Windows ME (2000), Windows XP (2001), Windows Server 2003 (2003), Windows Vista (2007),

Windows Server 2008 (2008), Windows 7 (2009), Windows 8 (2012) a Windows 10 (2015).

**JavaOS**

Vyvinut společností Sun Microsystems v roce 1997. Primárně vyvinut použitím programovacího jazyka Java. Byl vytvořen pro použití na jakémkoliv zařízení včetně osobních počítačů.

**MacOS X Server 1.0**

Vyvinut společností Apple Computer, Inc v roce 1999. Byl to předchůdce desktopové verze MacOS X 10.0 (2001), ve kterém se dramaticky změnilo grafické uživatelské rozhraní. MacOS X Server 1.0 byl vydán pro populární Apple Macintosh PC.[1]

**Android OS**

Varianta Linuxu vyvinuta společností Google v roce 2008. Operační systém zaměřen pro mobilní telefony a tablety. Obsahuje balíky pro většinu komponent, které mohou mobilní telefony obsahovat. Všechny aplikace jsou napsány v jazyce Java, spouštěné na JVM zvané Dalvik. Tato vlastnost zajišťuje nezávislost na platformě procesoru.[8]

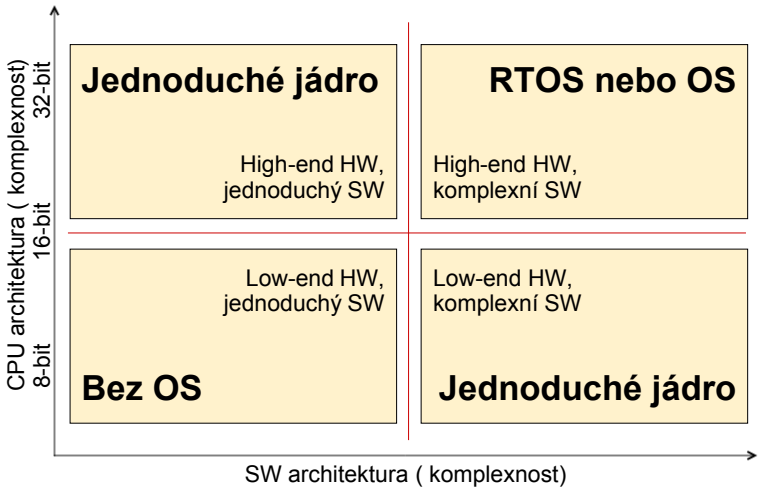
Operační systémy nevznikaly v historii pouze pro osobní počítače. S minimalizací čipů a vývojem vestavěných systému se také vyvíjely operační systémy pro dedikovaná zařízení za účelem řízení konkrétního zařízení. Tyto operační systémy se liší tím, že zařízení jsou většinou jednoúčelová a operační systém odpovídá konkrétním potřebám použitého hardwaru.

## 1.2 Druhy operačních systémů

Při výběru operačního systému pro osobní počítač záleží především na vkusu uživatele který si vybírá mezi Windows, Apple a nebo Linux. Pro dedikovaná zařízení už to tak jednoduché není. Je zde mnoho možností, které odrážejí rozmanitost použití jednotlivých systémů.

Otázkou je, zda je operační systém skutečně nutný. V dnešní době je vzácné najít vestavěný systém bez operačního systému. Pouze nejjednodušší zařízení mohou být sestaveny bez jádra nějakého druhu operačního systému. Celé spektrum dedikovaných zařízení může být reprezentováno následujícím grafem složitosti procesoru.

Graf 1.1 je rozdělen do čtyř kvadrantů. Pravý horní kvadrant představuje komplexní software na komplexním hardwaru. Tradiční provincie real-time operačních systémů a dalších složitých systémů. Na méně výkonném CPU může být stále vhodné nasazení základního jádra operačního systému, jestliže je software poměrně složitý. Někdy se ale používá výkonný hardware na kterém běží jednoduché aplikace, aby se




Obr. 1.1: Graf komplexnosti SW podle komplexnosti HW

dosáhlo požadované rychlosti běhu programu. V tomto případě nemusí být nezbytně požadované jádro operačního systému, ale jeho přítomnost zlepšuje škálovatelnost softwarové architektury a připravuje se na další zvyšování komplexnosti. Ve skutečnosti pouze jestliže má běžet jednoduchý SW na jednoduchém HW, není potřebný žádný druh jádra operačního systému.

Jestliže jsme dospěli k závěru že je pro náš projekt nezbytný operační systém, nastává otázka výběru samotného systému. Obecně existují čtyři možnosti: výběr high-end operačního systému jako je Linux nebo speciální verze systému Windows pro dedikovaná zařízení, výběr real-time operačního systému, nasazení jednoho z volně šiřitelných operačních systémů, nebo implementace vlastního jádra.

### 1.2.1 Komerční operační systémy

Na trhu existuje spousta komerčních operačních systémů pro dedikovaná zařízení. Výhodou komerčních systémů je nespočet dostupných RTOS produktů a mnoho z nich je od osvědčených renomovaných dodavatelů, ale tento výběr by měl být pečlivě zvážen. Velikost společnosti, zralost produktu a zázemí pro uživatele jsou velmi důležité faktory. Klíčovým požadavkem je technická podpora.

Při výběru RTOS systému vzniká dlouhodobý závazek mezi kupujícím a prodejcem softwaru. Jedním aspektem tohoto vztahu je posouzení možné migrace CPU v budoucnu. U kvalitního prodejce RTOS systému se lze dožadovat včasné podpory nových zařízení. Jejich produkt bývá již připraven na zjednodušení procesu portování na jiná zařízení.

Z technického hlediska je nevýhodou rozličnost systémů. Každé dedikované zařízení může být jiné - jiný procesor, paměť a různé periferie. Na komerčních systémech se jedná zejména o cenu, tedy od vybavení systému se odvíjí cena operačního systému, který má být v zařízení použit. Různé periferie mohou totiž spadat pod různé licence tvůrců operačních systémů a ne vždy se toto vyplatí.

### 1.2.2 Volně šiřitelné operační systémy

Volně šiřitelné operační systémy jsou většinou již hotové balíky, které si uživatel sice zakoupí, ale poté si systém upravuje dle vlastních potřeb. Hlavním lákadlem jsou nízké počáteční náklady a také uživateli nehrozí licenční poplatky.

Balíky s operačním systémem obsahují zdrojové kódy, které slouží jako reference, jelikož dokumentace bývá omezená a zřídka kdy existuje technická podpora.

Díky rozšířenosti SW mezi uživateli existují internetové komunity, odkud vývojáři mohou zdarma čerpat a přispívat vlastními zkušenostmi. Ačkoliv je to velká a lákavá výhoda, lidé se zpravidla zajímají o vývoj aktuálních verzí softwaru. Pokud tedy uživatel začínající se starší verzí softwaru potřebuje poradit, může narazit na problém s vyhledáním pomoci.

### 1.2.3 Vlastní operační systém

Výhodou tvorby jádra vlastního operačního systému je vlastní kontrola nad kompletním kódem. Předpokladem jsou ovšem vývojáři s příslušnými znalostmi. Nejsou zde žádné licenční náklady na vyvinutý kód, ale narůstají náklady na samotnou údržbu systému. Další výhodou sestavení vlastního jádra je vytvoření systému na míru odpovídajícím požadavkům projektu.[7]

### 1.2.4 Příklady operačních systémů pro dedikovaná zařízení

- FreeRTOS je třídou RTOS, tedy operační systém běžící v reálném čase, který je navržený dostatečně malý pro spuštění na mikrokontrolérech. Je to volně šiřitelný operační systém podporující 35 architektur. Je vyvíjen profesionály a striktně kvalitativně řízen. Vynálezcem FreeRTOS je Richard Barry. Dnes je vlastníkem a vývojářem společnost Real Time Engineers Ltd. FreeRTOS je zaměřen na omezená zařízení, pro které jsou jiné operační systémy příliš náročné. Jeho kernel je sepsán pouze ve třech zdrojových souborech, obraz jádra systému zabere přibližně od šesti do dvanácti kilobytů paměti. Je určen pro všechna cílová zaměření.[11]

- OpenWrt je popisován jako distribuce Linuxu pro dedikovaná zařízení. Místo vytváření statického firmware, OpenWrt nabízí plně přístupný souborový systém se správcem doplňků. Použití OpenWrt je typicky pro bezdrátové routery. Na systému pracuje spousta lidí, z nichž dva, Gerry Rozema a Mike Baker, jsou zodpovědní za jeho vývoj. Projekt se začal vyvíjet v roce 2004[10]
- Speciálním příkladem je Yocto Projekt, který nabízí uživateli prostředky k sestavení vlastní distribuce operačního systému založeném na systému Linux. Jedná se o volně šiřitelný projekt přinášející zázemí pro vývojáře operačních systémů s podporou jak samotných vývojářů projektu, tak s dalšími vývojáři operačních systémů pracující s projektem.[9]

## 1.3 Projekt Yocto

Yocto Projekt je kolaborativní open source projekt přinášející šablony, nástroje a metody pro vytvoření vlastního systému založeném na Linuxu pro dedikovaná zařízení bez ohledu na architekturu hardwaru. Yocto Projekt byl založen v roce 2010 jako společný projekt mnoha výrobců hardwaru, prodejců open-source operačních systémů a elektronických společností za účelem přinést pořádek do chaotického vývoje Linuxu pro dedikovaná zařízení.

Yocto Projekt jakožto open-source projekt pracuje s hierarchickou strukturou řízení založené na meritokracii spravované jeho hlavním architektem Richardem Purdiem, spolupracovníkem Linux Foundation. Tento systém umožňuje projektu zůstat nezávislý na komkoliv z členských organizací, kteří se podílejí různými způsoby na projektu a kteří projektu poskytují prostředky.

Yocto Projekt je kompletní vývojové prostředí pro vývoj Linuxových distribucí pro dedikovaná zařízení s nástroji, metadaty a dokumentací. Projekt využívá sestavovací systém na základě projektu OpenEmbedded, který používá nástroj BitBake k sestrojení kompletního Linuxového balíčku. Kombinace komponent BitBake a OpenEmbedded tvoří referenční sestavovací systém, historicky známý jako Poky. [9]

# 2 MEZI-PROCESOROVÁ KOMUNIKACE

Moderní jednočipové systémy mívají více procesorových jader různých druhů. takovýmto kombinacím se říká Asymetrická Multiprocesorová konfigurace (AMP). Každé procesorové jádro přitom může spouštět různé instance operačních systémů, jako High-Level operační systémy, například Linux, nebo operační systémy reálného času, případně může spouštět vlastní firmware.

Podle potřeb se do jednočipových systémů implementují různé kombinace více výkonných procesorových jader s méně výkonnými. Vznikají tedy kombinace asymetrických a symetrických multiprocesorových konfigurací. Například jednočipový systém OMAP společnosti Texas Instruments (Open Multimedia Applications Platform), konkrétně OMAP verze 4 je osazen duálním procesorem Cortex-A9, duálním procesorem Cortex-M3 a jedním C64x+ jádrem. Typicky je pak na duálním procesoru Cortex-A9 spouštěn operační systém Linux v režimu symetrické multiprocesorové konfigurace a všechny ostatní jádra spouštějí jejich vlastní instance operačního systému reálného času a zavaděče systému BIOS v režimu asymetrické konfigurace.

Asymetrické multiprocesorové konfigurace většinou slouží k ovládání hardwarových akcelerátorů a tím pádem k odlehčení zátěže výkonných procesorových jednotek. Tyto procesory mohou rovněž ovládat časové závislé senzory, pro řízení libovolného bloku hardwaru případně spouštět úlohy v pozadí zatímco je hlavní procesor v úsporném režimu.

Pro použití multiprocesorových konfigurací minimálně se dvěma procesorovými jádry je potřeba zajištění správy řízení a komunikační framework. Správa řízení je zpravidla nezávislá na komunikačním frameworku a je zodpovědná pouze za řízení stavu jader. Komunikační framework naopak vyžaduje závislost sekundárního jádra, ačkoliv samotný komunikační protokol je nezávislý.[2]

## 2.1 MCC

Více-jádrová komunikace MCC (Multi-Core Communication) je původně vyvinutá knihovna společností Freescale pro komunikaci mezi heterogenními jádry. Jedná se o softwarový mechanismus využívající výhody hardwarové synchronizace a komunikačních možností. Je vyvinuta pro jednoduchou a rychlou metodu komunikace mezi jádry s možností implementace pro jednoduché aplikace až pro využití v prostředí operačního systému Linux. Využívá sdílenou paměť RAM, přerušení a hardwarové semafory. Lze konfigurovat velikost vyrovnávací paměti, jejich počet a počet koncových bodů. Volání aplikačního rozhraní jsou jednoduché odesílání a příjem mezi koncovými body. Koncové body jsou součásti návrhu uživatelského systému - není

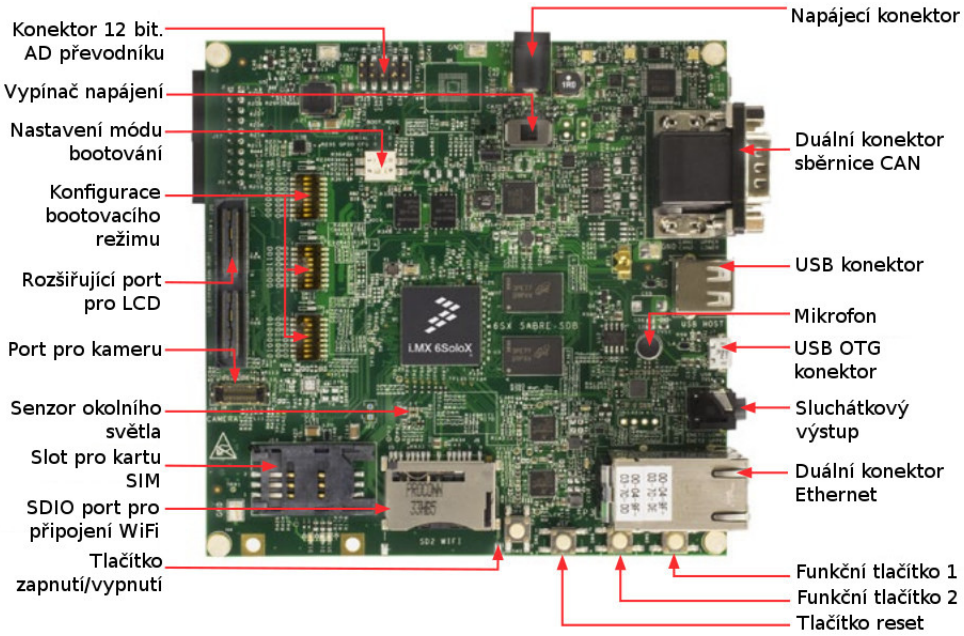
zde žádný vyhledávací protokol. MCC je podobný internetovému protokolu, který adresuje koncové body pomocí IP adres, protokolu (TCP nebo UDP) a čísla portu. Datové přenosy vznikají v pevně stanoveném zásobníku paměti v RAM. Všechny zásobníky jsou alokovány během inicializace a pouze jeden zásobník je sdílený mezi všemi jádry. Odesílání a příjem dat probíhá pouhým čtením a zápisem do tohoto zásobníku. Zásobník je potřeba uvolňovat po každém přenosu dat. Přístup do zásobníku je řízen hardwarovými semafory. Pro použití s operačním systémem Linux je potřeba zavést aplikační rozhraní do kernelu systému.[6]

## 2.2 RPMSG

RPMSG - Remote Processor Messaging framework je mezi-jádrový komunikační protokol, narozdíl od MCC již implementovaný v kernelu operačního systému Linux. RPMSG používá adresování koncových bodů podobné jako MCC s tím rozdílem, že využívá 32 bitová celá kladná čísla. Adresy identifikují obsluhu přijímače konkrétního ovladače. Každé zařízení je jako komunikační kanál s vlastní zdrojovou a cílovou adresou. RPMSG implementuje koncové body jako strukturu. Na rozdíl od MCC obsahuje rpmsg adresu a interní implementační struktury.[6]

RPMSG je založen na virtuální komunikační sběrnici, jež umožňuje ovladači v kernelu komunikovat s ostatními jádry v systému. Ovladače mohou být zavedeny jako konkrétní uživatelské rozhraní pokud to je uživatelem vyžadováno. Prakticky se využívá pouze jedna registrace každého fyzického jádra v systému, ale nejsou zde žádná omezení registrovat jádro vícekrát.[2]

# 3 MIKROKONTROLÉRY I.MX

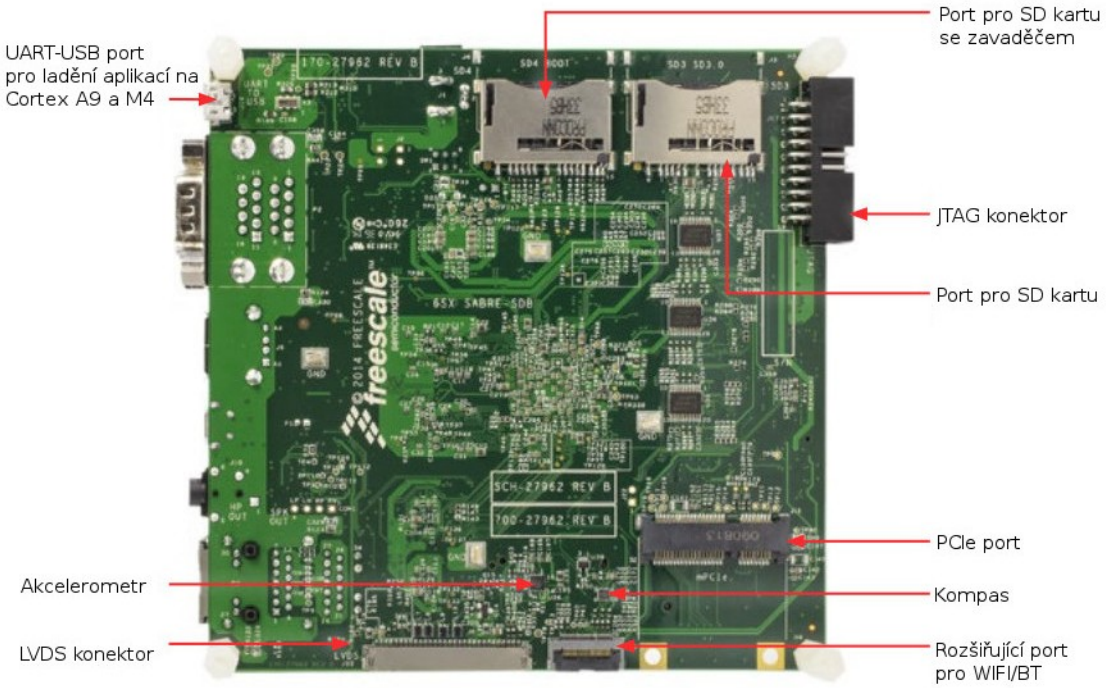



Obr. 3.1: Horní strana vývojové desky SabreSD.

Mikrokontroléry jsou jednočipové systémy obsahující periferie pro různé typy zařízení. Rozdíl oproti počítači je především v tom, že u počítače bývají periferie osazeny na základní desce, která obsahuje řadič, tzv. severní a jižní můstek, který tyto periferie obsluhuje a informace posílá do hlavní procesorové jednotky na zpracování. Zatímco mikrokontrolér obsahuje tyto periferie již v jednom společném čipu s procesorem. Proto se používá také název SoC (System on Chip) - jednočipové systémy. Pro použití těchto periferií je potřeba určit na jaký výstup z mikrokontroléru budou vyvedeny potřebné kontakty jednotlivých periferií a k těmto kontaktům přidat podpůrné obvody pro vyvedení na případný konektor.

Mikrokontroléry řady i.MX jsou proprietární mikrokontroléry pro multimediální aplikace založené na architektuře ARM se zaměřením na nízkou energetickou spotřebu a více-jádrová systémová jednočipová řešení. Mikrokontroléry řady i.MX mají široké uplatnění v automobilovém i uživatelském spotřebním průmyslu.

Dnes je známo již celkem 7 řad mikrokontrolérů z rodiny i.MX. Jako první byla vydána řada i.MX 1 na přelomu roku 2001/2002 založena na architektuře ARM920T s pracovním kmitočtem 100 až 200 MHz. O deset let později v roce 2011 byly představeny první mikrokontroléry postavené na platformě ARM Cortex A9 ve verzích jednojádrových, dvoujádrových a čtyřjádrových jednočipových systémů v řadě i.MX 6. Mikrokontroléry pracují na kmitočtu 528 až 1200 MHz. Vybrané modely jsou




Obr. 3.2: Spodní strana vývojové desky SabreSD.

vybaveny výkonnými grafickými čipy Vivante s podporou dekodéru FullHD videa. Stejně jako v předchozích řadách jsou mikrokontroléry osazeny bezpečnostními hardwarovými akcelerátory.[3]

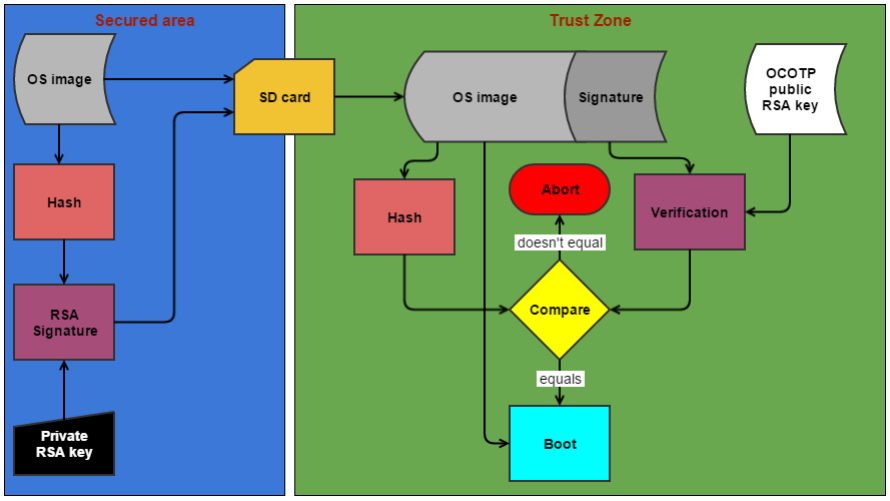
## 3.1 Hardwarové specifikace

Pro tuto práci byl použit mikrokontrolér i.MX 6SoloX [3.1, 3.2], jež je vybaven jedním výkonným procesorovým jádrem Cortex A9 s pracovním kmitočtem 1GHz a jedním nízko-energetickým jádrem Cortex M4 s pracovní frekvencí 227MHz. Na čipu se nachází také grafické jádro Vivante, hardwarové zabezpečovací akcelerátory, rozhraní LVDS, HDMI, USB 2.0, Ethernet, PCIe, S-ATA a další. Čip je osazen na vývojové desce SABRE SDB s podpořeným vývojem OS Linux a Android. Na desce se dále nachází paměť RAM velikosti 1GB typu DDR3L s pracovní frekvencí 400MHz, 2x32MB QSPI NOR flash paměť, 3 sloty pro SD kartu a další. K mikrokontroléru jsou připojeny senzory jako 3-osý akcelerometr, digitální kompas a světelný senzor. Deska umožňuje nastavení několika způsobu načtení obslužných systému jako například načtení zavaděče z SD karty, paměti flash nebo přes sériovou sběrnici USB.[5]

## 3.2 Možnosti systémového zabezpečení

Bezpečnost je běžný požadavek pro platformy postavené za pomocí i.MX mikrokontrolérů, ačkoliv specifické potřeby se mění podle platformy a trhu. Typ a cena chráněného majetku na přenosném uživatelském zařízení se velice liší od zařízení použitých v automobilních a průmyslových platformách. To samé platí pro útoky a úroveň nebezpečí ohrožujících tento majetek. Návrhář platformy musí zvolit náležitou sestavu protiopatření, aby dosáhl relevantní potřeby ochrany platformy.

I.MX 6SoloX zahrnuje škálu bezpečnostních opatření pro vývojáře, která mohou být použity jednotlivě nebo kombinovaně k podpoře bezpečnosti architektury navrhované platformy. Tyto prvky jsou navrženy pro vzájemnou spolupráci a mohou být integrovány společně s odpovídajícím softwarem pro vytvoření požadované úrovně ochrany. Součástí ochranných prvků je i.MX 6SoloX doplněn pro obecný účel akcelerátory ke zvýšení výkonnosti vybraných kryptografických algoritmů průmyslových standardů. I.MX 6SoloX nabízí tyto bezpečnostní komponenty:




Obr. 3.3: Blokové schéma funkce HAB.

### 3.2.1 HAB

[3.3] High Assurance Boot, neboli zabezpečení zavaděče je prvek v systémovém zavaděči, verifikující integritu načítaného systému. Využívá politiku asymetrické kryptografie algoritmu RSA. HAB detekuje autorizaci zaváděného systému a brání načtení neautorizovaného, případně modifikovaného systému během bootovací sekvence. HAB používá digitální podpis o délce maximálně 4096 bitů. Je zaváděn v tzv. Trust Zóně, aby nedošlo k účelnému podstrčení neautorizovaného klíče. Ty jsou uloženy v OTP paměti a po zápisu klíče do této paměti jej již nelze přepsat. V paměti

je uložena veřejná část klíče, kterou se ověřuje pravost dat podepsaných soukromým klíčem u vydavatele softwaru.

### 3.2.2 SNVS

Secure Non Volatile Storage neboli zabezpečení energeticky nezávislého úložiště, v kombinaci s HABem a obvody pro detekci zásahu do systému definuje stav zabezpečení a umožňuje přístup do paměti s uloženými šifrovacími klíči. V případě, že dojde k narušení, vyšle oznámení systému a provede předem nadefinovaný proces pro znemožnění přístupu narušitele do paměti. Součástí SNVS je také paměť sloužící pro dočasné uložení citlivých informací, jež se při jakémkoliv narušení bezpečnosti okamžitě smaže.

### 3.2.3 TrustZone

TrustZone neboli bezpečnostní zóna umožňuje privilegovaný přístup k vybraným periferiím včetné pamětí. TrustZone definuje bezpečné prostředí pro kritický software. Chrání jej proti softwarovým útokům včetně aplikací, službám a ovladačům. TrustZone zavádí uživatelské, administrační a další privilegované módy v nichž je možno provádět nadefinované operace s omezeným přístupem v systému.

### 3.2.4 OCRAM

Secured On-Chip RAM je vestavěná paměť RAM s ochranou vybrané zóny pomocí OCRAM ovladače. Do definované paměti lze zapisovat zabezpečeně nebo nezabezpečeně v závislosti na aktuální konfiguraci Trust Zóny. Naslouchá Centrální bezpečnostní jednotce a řídí přístup do paměti definované adresou počátku. Paměť je vždy zabezpečena od této adresy do poslední adresy paměti OCRAM.

### 3.2.5 OCOTP

On-Chip One Time Programable memory je zabudovaná jednou programovatelná paměť se zabudovanými elektrickými pojistkami. Po zápisu do této paměti ji již nelze modifikovat. Slouží pro ukládání šifrovacích, dešifrovacích a certifikačních klíčů. Přístup do paměti je podmíněn konfigurací Centrální zabezpečovací jednotky a dle konfigurace bezpečnostního protokolu také dalšími bezpečnostními prvky systému.

### 3.2.6 CSU

Central Security Unit je Centrální zabezpečovací jednotka sloužící k řízení jednotlivých bezpečnostních prvků systému. Je důležité, jednotku korektně nakonfigurovat, aby nedošlo k nechtěnému narušení bezpečnosti.

### 3.2.7 RDC

Resource Domain Controller - řadiče doménových prostředků řídí přístup k periferiím a paměti pro jednotlivá jádra. Izoluje prostředky pro vybrané procesorové jádro tak, aby žádné jiné jádro nemohlo k těmto prostředkům přistupovat. RDC podporuje doménové priority, takže doména s vyšší prioritou bude mít vybrané prostředky vždy k dispozici.

### 3.2.8 SJC

System JTAG Controller - Systémový ovladač JTAG. JTAG port umožňuje přístup k ladění hardwarových bloků včetně procesoru ARM a systémové sběrnice. Dovoluje programové řízení, manipulaci a přístup k periferiím čipu a paměti. JTAG port musí být přístupný při vývoji, testování a řešení problémů. Vzhledem k jeho schopnostem přináší bezpečnostní hrozbu pro přístup k citlivým datům. Proto je tato funkce doplněna o možnost podmíněného přístupu na čtyřech úrovních, mezi které patří autorizovaný přístup pomocí tajného klíče.

### 3.2.9 SDMA

Smart Direct Memory Access, neboli tzv. chytré řízení přímého přístupu do paměti přináší ovladač řídící přístup do paměti za předem stanovených podmínek definovanými módy uzamčení.

### 3.2.10 DryICE

Jedná se o monitorování frekvence, teploty a napětí v reálném čase.

### 3.2.11 CAAM

Cryptographic Acceleration and Assurance Module přináší akcelerátory pro kryptografické operace, které jsou výpočetně velmi náročné. Obsahuje šifrovací a hashovací funkce, generátor pseudonáhodných čísel certifikovaný Národní Institucí Standardů a Technologií (NIST).[4]

### 3.2.12 Softwarové metody zabezpečení

Mimo metody zabezpečení s hardwarovými akcelerátory jsou k dispozici také softwarové metody. Jsou to algoritmy vyvíjené pro nasazení do systémů bez možností hardwarových akcelerátorů. Softwarovými knihovnami se řeší zabezpečení komunikace mezi jednotlivými koncovými uzly.

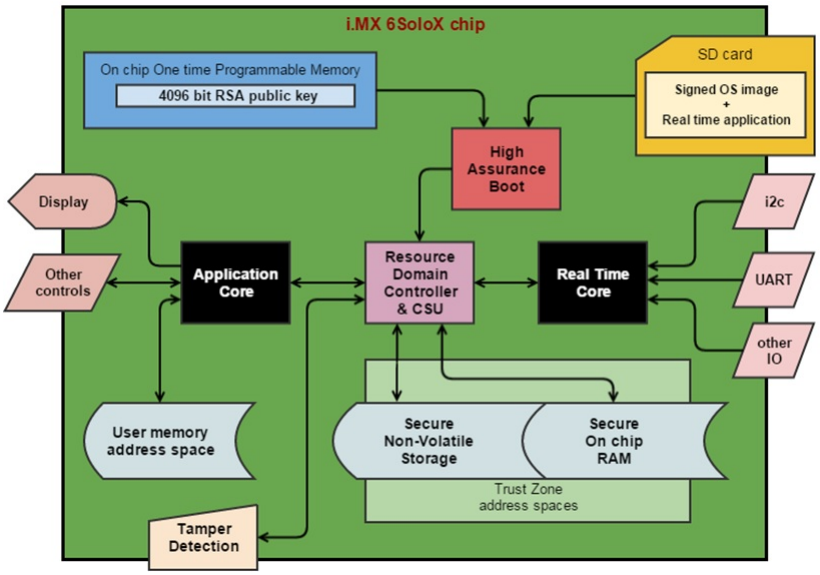
Mimo samotné zabezpečení přenosu dat šifrováním se řeší také problém předání šifrovacího klíče, neboť zjištěním šifrovacího klíče cizí stranou znehodnocuje samotný účel šifrování. Byly proto navrhnuty algoritmy pro výměnu klíčů, mezi nejznámější patří algoritmus Diffie-Hellmana. Ten je založen na asymetrické kryptografii, kdy se generují dva klíče, ze kterých se jeden uveřejní a druhý, soukromý klíč, se dále aplikuje na vytvoření společného symetrického klíče. Po výměně klíče lze použít dostupné symetrické algoritmy pro zabezpečení komunikace. Použití algoritmu Diffie-Hellmana bude dále popsáno v praktické části diplomové práce.

# 4 NÁVRH SYSTÉMU

Po seznámení s jednočipovými systémy rodiny i.MX a možnostmi meziprocesorové komunikace, které jsou pro tento čip k dispozici, byl vybrán mikrokontrolér i.MX 6SoloX především díky dostupným prostředkům pro implementaci meziprocesorové komunikace, širší podpoře veřejné komunity a dostupné dokumentaci. Protokol pro meziprocesorovou komunikaci byl zvolen RPMSG díky implementaci v systému Linux a dostupnými informacemi o použití RPMSG pro tento čip, včetně demonstračních aplikací.

## 4.1 Hardwarové řešení zabezpečení

Dle předchozích zjištění je vhodné implementovat bezpečnostní hardwarové akcelerátory, kterými je jednočipový systém vybaven. Je nutné zvolit takovou kombinaci prvků, která by odpovídala požadované úrovni zabezpečení. Podle vlastností jednotlivých bloků a potřeb pro zabezpečení mezi-procesorové komunikace byl sestaven návrh řešení sestávající z vybraných bloků.[4.1]



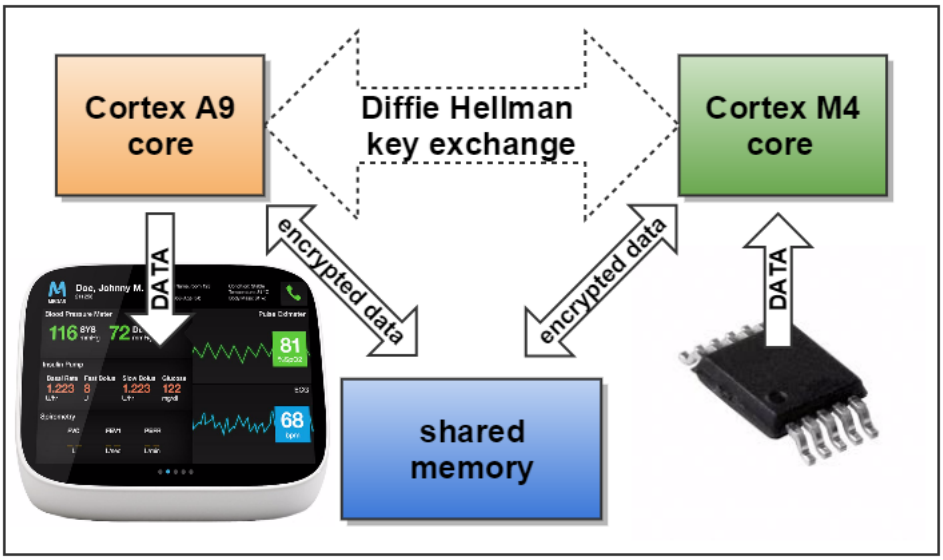

Obr. 4.1: Blokové schéma návrhu zabezpečení meziprocesorové komunikace.

V tomto návrhu se uvažuje s několika zabezpečovacími prvky. Prvním v řadě je využití HABu, kde se z páru asymetrických klíčů podepíše obraz systému soukromým klíčem a veřejný se zapíše do OTP paměti. Nepodepisuje se celý obraz systému, ale pouze ta část, kde nechceme uživateli dovolit modifikaci dat. Tímto zamezíme načtení libovolného systému, který by mohl být schopen zasahovat do systému. Dále

je zde implementován RDC, pro zamezení přístupu jednotlivých jader k periferiím, určeným výhradně pro konkrétní jádro. Komunikace RPMSG probíhá čtením a zápisem do vyhrazené paměti, proto je zde vymezený prostor v zabezpečené oblasti paměti RAM. SNVS zde slouží pro případné uložení tzv. Master Key, tedy hlavního klíče pro šifrování přenášených dat. Tato paměť zabrání vyčtení klíče jeho okamžitým vymazáním. A to v případě jestliže detektor narušení ohlásí nedovolený zásah do softwaru či hardwaru systému sundáním krytu přístroje, podezřelé kolísání teploty, případně další porušení ochran připojených k detektoru.

K realizaci tohoto návrhu však výrobce nedává k dispozici konkrétní informace nutné pro konfiguraci jednotlivých částí zabezpečovacího systému. Jelikož se jedná o velice citlivé údaje, realizaci zabezpečení systému vývojář provádí za přísných podmínek udržení utajení a po podpisu smlouvy o mlčenlivosti.

## 4.2 Softwarové řešení zabezpečení




Obr. 4.2: Blokové schéma zabezpečení pomocí algoritmu DH.

Pro tuto práci byl tedy zvolen softwarový způsob zabezpečení komunikace mezi jádry. Pro implementaci zabezpečení byla zvolena softwarová knihovna CycloneSSL, která byla vyvinuta pro použití ve vestavěných systémech. K dispozici jsou ukázkové aplikace, mimo jiné i pro mikrokontroléry společnosti NXP. Součástí knihovny jsou aplikace pro webové služby jako FTP, DHCP, DNS, HTTP, PPP, SNMP atd. Pro účel zabezpečení komunikace mezi jádry byla použita pouze část knihovny realizující

předání šifrovacího klíče pomocí algoritmu Diffie Hellman a poté samotné symetrické šifrování přenášených dat. [4.2]

Základní myšlenkou bylo vytvoření zabezpečeného kanálu, který by i přes možnost vyčtení dat z paměti nebylo možné zneužít, případně podsunout data vlastní. Je tedy potřeba implementovat knihovnu, jak pro aplikaci v jádře Cortex M4, tak i pro jádro Cortex A9, na kterém běží vygenerovaná Linuxová distribuce projektem Yocto. Pro demonstraci zabezpečeného přenosu dat je v jádře Cortex M4 prováděno čtení dat ze senzoru akcelerometru, data jsou poté zašifrována a odeslána protokolem RPMSG. Předání klíče probíhá jednorázově po každém spuštění obou jader. Na výkonnějším jádře je spuštěn webový server demonstrující zabezpečený přenos.

# 5 REALIZACE SYSTÉMU

V praktické části bude vysvětlen postup implementace zabezpečení komunikace mezi jádry, začínaje sestavením Linuxové distribuce pro mikrokontrolér i.MX 6SoloX, implementace softwarové knihovny pro výměnu symetrického klíče a vytvoření demonstrační aplikace. Dále bude představena aplikace pro obsluhu akcelerometru, implementace softwarové knihovny pro jádro Cortex M4 a představení aplikačního rozhraní pro RPMSG v operačním systému FreeRTOS.

## 5.1 Yocto

Sestavení operačního systémuneboli kompilace jádra je velice náročný proces vyžadující vysoký výpočetní výkon. Podle dokumentace Yocto jsou minimální požadavky 1 GB systémové paměti a nejméně 32 GB prostoru na disku. Tyto parametry se však odvíjí od komponent,které má výsledný systém obsahovat. Pro sestavení linuxové distribuce je proto vhodné zvolit stroj s vyšším výpočetním výkonem a operační systém, který je s projektem Yocto kompatibilní.

### 5.1.1 Zdrojové kódy

Pro práci s projektem Yocto je potřeba nejprve nainstalovat potřebné základní balíčky pro podporovanou distribuci Ubuntu.
Balíčky lze získat například následujícím příkazem :

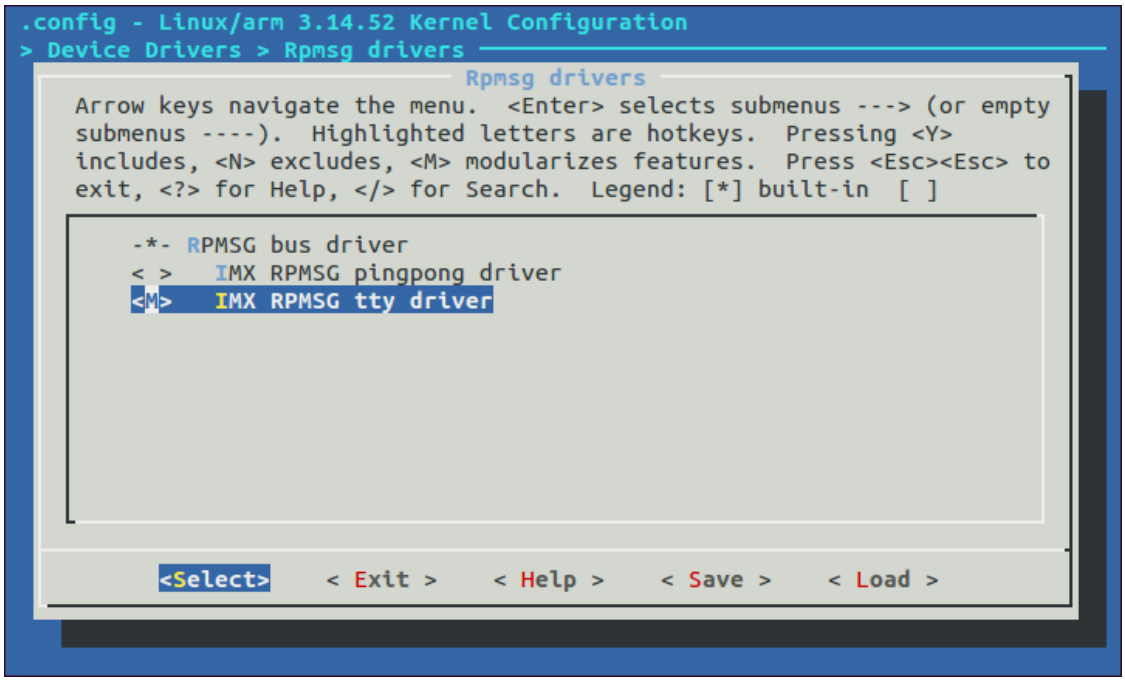
```
$ sudo apt-get install gawk wget git-core diffstat unzip \
  texinfo build-essential chrpath socat libsdl1.2-dev \
  xterm curl libncurses5-dev
```

Abychom mohli pracovat s více verzemi Yocta, je vhodné doinstalovat nástroj Repo:

```
$ mkdir ~/bin
$ curl http://commondatastorage.googleapis.com/\
  git-repo-downloads/repo > ~/bin/repo
$ PATH=$PATH:~/bin
$ chmod a+x ~/bin/repo
```

Poté je potřeba do systému stáhnout samotné zdrojové kódy projektu Yocto. Ty se dají získat přímo u výrobce s podporou našeho konkrétního hardwaru.
Repozitář získáme následujícími příkazy :

```
$ repo init -u git://git.freescale.com/imx/\
  fsl-arm-yocto-bsp.git -b imx-3.14.52-1.1.0_ga
$ repo sync
```




Obr. 5.1: Přidání ovladače RPMSG do kernelu systému

Určíme si cílovou složku pro sestavení systému nadefinováním systémové proměnné:

```
$ MACHINE=imx6sxsabresd source fsl-setup-release.sh -b build
```

A přidáme moduly s ovladači pro práci s RPMSG jako s virtuálním terminálem pomocí interaktivního nástroje pro úpravu kernelu menuconfig:

```
$ bitbake -c menuconfig linux-imx
```

V menu ovladačů zařízení vybereme RPMSG a označíme IMX RPMSG tty driver pro virtuální terminál.[5.1]

Nyní máme na výběr cílové sestavy rozdělené podle implementovaných součástí jako grafické rozhraní, podporu pro aplikace Qt5, sady nástrojů a vývojové prvky. Pro naše účely postačí minimální konfigurace core-image-minimal s menšími úpravami konfigurace. Protože budeme potřebovat doinstalovat do systému moduly pro Linux i pro Python, upravíme konfigurační soubor core-image-minimal.bb umístěném v adresáři 'sources/poky/recipes-core/images/'. Do souboru přidáme potřebné moduly, které se skompilují společně s distribucí. Jedná se především o tyto balíčky:

- vim - oblíbený textový editor pro příkazovou řádku, který graficky formátuje kód podle programovacího jazyka
- python-pip - jednoduchý instalátor modulů pro Python s nastavenými repozitáři zdrojových kódů. Představuje zjednodušení instalace modulů a především také jejich závislostí jednoduchým zadáním příkazu pip install <modul>.

- binutils - kolekce binárních nástrojů pro manipulaci s objektovým kódem při vývoji softwaru.
- cpp - makro procesor používaný automaticky kompilátorem zdrojových kódů jazyka C k transformaci programu před samotnou kompilací.
- g++ - kompilátor C++ kódů
- make - utilita pro automatizaci překladu zdrojových kódů do binárních souborů.

Tyto balíčky společně s pomocnými doplňky a knihovnami jako cpp-symlinks, gcc-symlinks, g++-symlinks, gettext, libstdc++, libstdc++-dev, file a coreutils nám umožní přidávání aplikací bez potřeby balíčkových manažerů. Tyto balíčky se připíšeme k ostatním do proměnné IMAGE_INSTALL.

Ve výchozím stavu je pro rootfs systému, tedy pro kořenový adresář vyhrazeno místo o velikosti 800 MB. Náš systém bude portován na SD kartu o velikosti 8 GB. Proto nastavíme velikost dostupné paměti pro systémový adresář přibližně 7 GB, tak aby zbylo ještě místo pro zavaděč a další části paměti potřebné pro zavedení a běh systému. To provedeme přidáním proměnné IMAGE_ROOTFS_SIZE ?= "7000000".

### 5.1.2 Sestavení OS

Při prvním sestavení systému se podle vybraných komponent a jejich závislostí stáhnou zdrojové kódy, které budou následně zkompilovány pro platformu mikrokontroléru i.MX 6SoloX s jádrem Cortex A9. Díky našemu výběru minimální konfigurace, která je bez grafického rozhraní a jiných pro nás nepotřebných modulů bude generování systému časově méně náročné.

Nyní, když máme vše připravené, můžeme se pokusit sestavit Linuxové jádro. Protože jsme použili repozitář Yocta výrobce čipu, máme zde vše připravené pro spuštění generování. Pro jistotu si však můžeme ověřit správnost konfigurace naší vybrané platformy. Ta se nachází v konfiguračním souboru conf/local.conf.
Soubor s konfigurací otevřeme například příkazem :

```
$ vim conf/local.conf
```

Samotný soubor s konfigurací poté může vypadat například takto :

```
MACHINE ??= 'imx6sxsabresd'
DISTRO ?= 'fsl-imx-fb'
PACKAGE_CLASSES ?= "package_rpm"
EXTRA_IMAGE_FEATURES = "debug-tweaks"
USER_CLASSES ?= "buildstats image-mklibs image-prelink"
PATCHRESOLVE = "noop"
```

```
BB_DISKMON_DIRS = "\
␣␣␣␣STOPTASKS,${TMPDIR},1G,100K␣\
␣␣␣␣STOPTASKS,${DL_DIR},1G,100K␣\
␣␣␣␣STOPTASKS,${SSTATE_DIR},1G,100K␣\
␣␣␣␣ABORT,${TMPDIR},100M,1K␣\
␣␣␣␣ABORT,${DL_DIR},100M,1K␣\
␣␣␣␣ABORT,${SSTATE_DIR},100M,1K"
PACKAGECONFIG_append_pn-qemu-native = "␣sdl"
PACKAGECONFIG_append_pn-nativesdk-qemu = "␣sdl"
ASSUME_PROVIDED += "libsdl-native"
CONF_VERSION = "1"

BB_NUMBER_THREADS = '4'
PARALLEL_MAKE = '-j 4'

DL_DIR ?= "${BSPDIR}/downloads/"
ACCEPT_FSL_EULA = "1"
```

Jak zde můžeme vidět, konfigurace je nastavena pro imx6sxsabresd, což odkazuje na konfigurační soubor s parametry vývojové desky SABRE SDB osazené mikrokontrolérem i.MX 6SoloX. Dále zvolená distribuce fsl-imx-fb (frame buffer graphics) odkazuje na konfigurační soubor bez grafického rozhraní pro mikrokontroléry řady i.MX. Další proměnné nastavují podporu balíčků rpm, ladící nástroje, nastavení uživatelských tříd, řízení monitoringu situace nedostatku místa v paměti atd. Dále jsou zde informace pro sestavovací skript ohledně využití přístroje, na kterém sestavování poběží, jako počet vláken, které se přiřadí procesoru a počet paralelně zpracovávaných úloh. Optimální nastavení je jedno vlákno pro jedno jádro procesoru při zpracování 4 úloh najednou, tedy proces bude mít k dispozici 4 jádra procesoru.

Nakonec jsou nastaveny proměnné pro určení místa, kam budou zdrojové soubory staženy a informace, zda již uživatel četl a souhlasil s podmínkami použití produktu společnosti NXP. Označení místa úložiště je důležité zachovat pokaždé stejné, protože při sestavování systému se kontroluje, zda již soubory nebyly staženy, aby příští sestavování mohlo proběhnout rychleji a nezaplňovala se zbytečně paměť duplicitními soubory.

Dalším krokem je spuštění generátoru. Pro náš případ budeme generovat obraz s přednastavenou konfigurací komponent neobsahující grafické rozhraní a základní prvky pro práci se systémem. Název konfiguračního souboru je core-image-minimal, který se zadává jako vstupní parametr. Doba generování takového obrazu je závislá na prostředcích počítače, na kterém sestavování běží. Čas generování se pohybuje

přibližně od 2 do 10 hodin. Při druhém pokusu generování již neprobíhá celková kompilace, ale pouze doplnění dle úprav výběru komponent systému, tedy použijí se již zkompilované součásti systému. Samotné znovu zabalení obrazu systému zabere přibližně 15 minut.
Generování spustíme příkazem :

```
$ bitbake core-image-minimal
```

Požadované místo pro generování je přibližně 40 GB.

### 5.1.3 Výsledky sestavení

Pro generování byl použit počítač s těmito parametry:

- Procesor Intel Core i5-4310m, 2 fyzická a 2 virtuální jádra s frekvencí 2,7 GHz.
- Dostupná systémová paměť RAM je 8 GB
- Pevný disk SSD Intel 530 Series - 180GB s výrobcem uvedenou rychlostí čtení 540 MB/s a zápisu 490 MB/s
- Operační Ubuntu 14.04 LTS

Výsledný čas sestavování obrazu systému je 3 hodiny a 5 minut. Vygenerované obrazy systému najdeme ve složce :

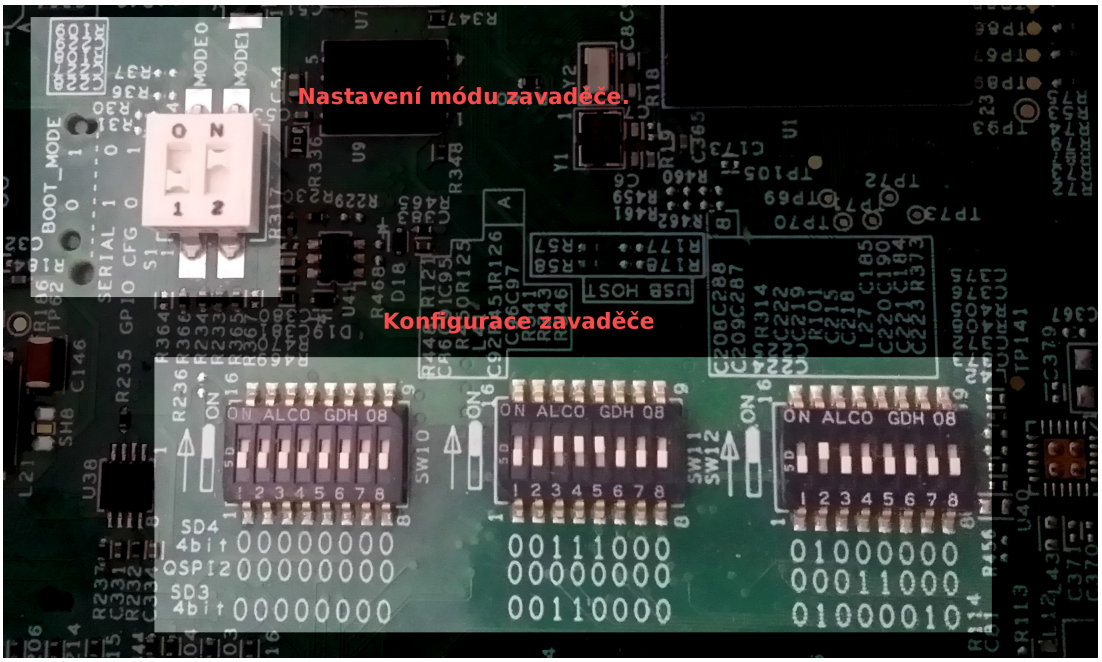
```
./build/tmp/deploy/images/imx6sxsabresd/
```

Další vygenerované soubory jsou například logy se záznamem postupu generování, stažené zdrojové soubory, které byly pro systém kompilovány a další pomocné balíky potřebné pro úspěšné sestavení systému. Obrazy sytému jsou generovány s názvem končícím číslem představující rok, měsíc, den a identifikační číslo procesu. Tyto soubory zde zůstávají jako předchozí verze systému, dokud je uživatel sám neodstraní. Mimo tyto soubory jsou zde také zástupci se zkráceným názvem bez identifikačního čísla, které odkazují vždy na poslední generovanou verzi systému. Jsou zde soubory jednotlivých částí systému, jako zavaděč uboot, rootfs, kernel a mimo tyto soubory je vygenerován kompletní obraz systému s přesným uspořádáním jednotlivých částí, kvůli nastavené adresaci v zavaděči. Ten lze přímo zkopírovat na SD kartu (v tomto případě /dev/sdb2), například pomocí příkazu:

```
$ sudo dd if=core-image-minimal-imx6sxsabresd.sdcard\
  of=/dev/sdb2
```

### 5.1.4 Spuštění systému a test

Po úspěšném generování je potřeba nastavit konfigurační přepínače na vývojové desce pro správné zavedení systému[5.2]. Tyto přepínače nám umožňují výběr zdroje a způsobu zavedení systému, jako například možnost zavedení z různých slotů pro




Obr. 5.2: Přepínače pro nastavení zavaděče.

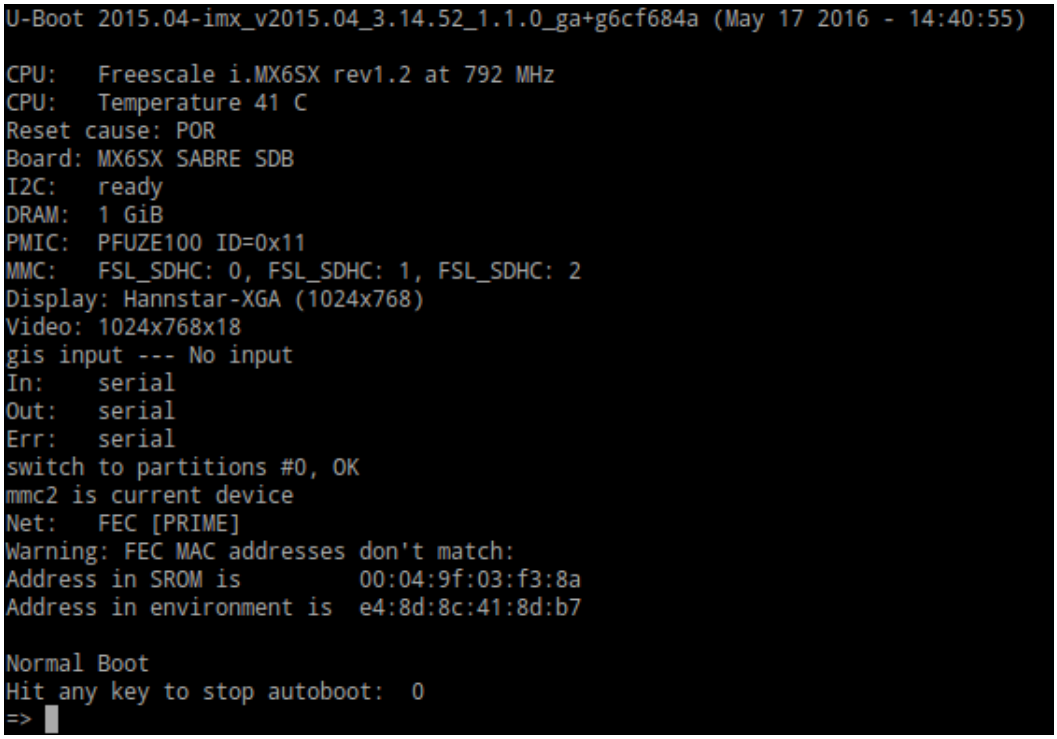
SD karty nebo přes sériovou sběrnici USB, v jiných případech lze nastavit zavedení z jiných sběrnic a pamětí jako eMMC, SATA, NAND, SPI-NOR a QuadSPI. V našem případě budeme zavádět operační systém ze slotu pro SD karty na pozici SD4, a proto nastavíme piny dle obrázku 5.2.

Po správném nastavení konfiguračních pinů již stačí vložit SD kartu s našim systémem do pozice SD4 a zapnout napájení. Na konzoli procesoru CorteX A9 připojené k ladícímu portu UART-USB můžeme pozorovat proces zavádění. Nejprve se spustí samotný zavaděč, v tomto případě U-Boot [5.3]. Tento zavaděč má přístup k veškeré dostupné paměti mikrokontroléru i SD karty. Nesprávný zásah do paměti může porušit náš systém, případně nechtěně změnit předešlá nastavení mikrokontroléru.

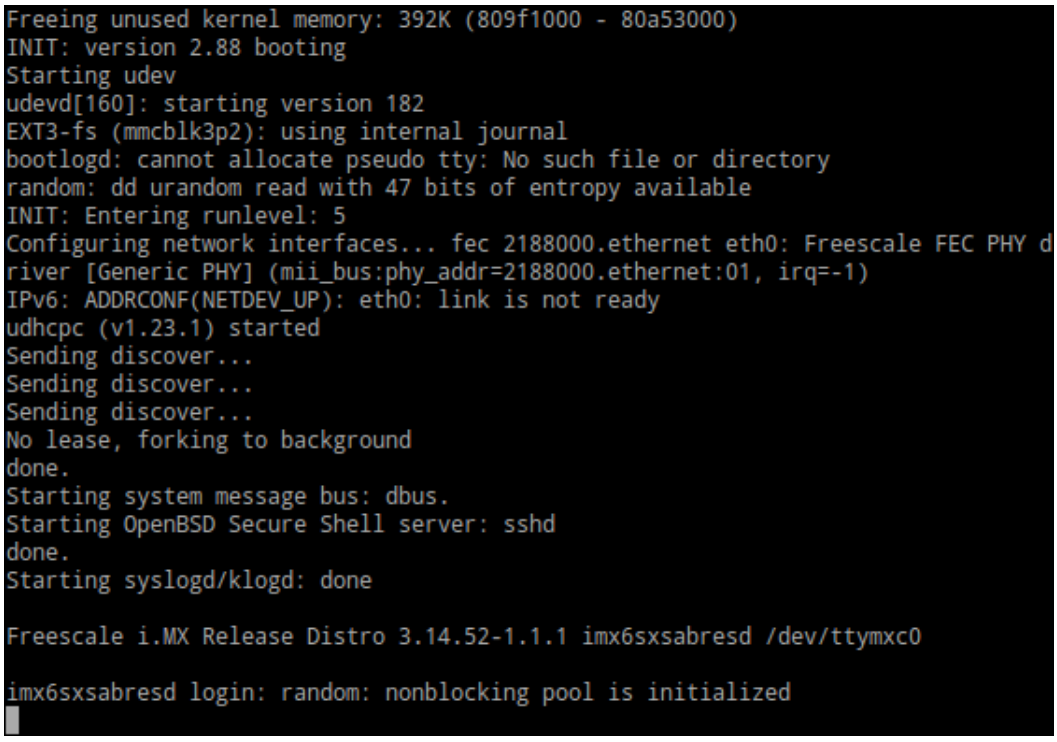
Po uplynutí odpočítávání U-Boot začne zavádět systém dle kritérií uvedených při generování systému. Po delším výpisu informací o prováděných instrukcích vypadá konzole načteného systému Linux podobně jako na obr.[5.4]. Do systému se přihlásíme jako root bez hesla, což je výchozí nastavení po vytvoření distribuce.

### 5.1.5 Kernel

Kernel neboli jádro operačního systému je část operačního systému, která je při startu zavedena do operační paměti a je jí předáno řízení. U pokročilých operačních systémů jádro nikdy neztrácí kontrolu nad počítačem a po celou dobu jeho běhu koordinuje činnost všech spuštěných procesů.

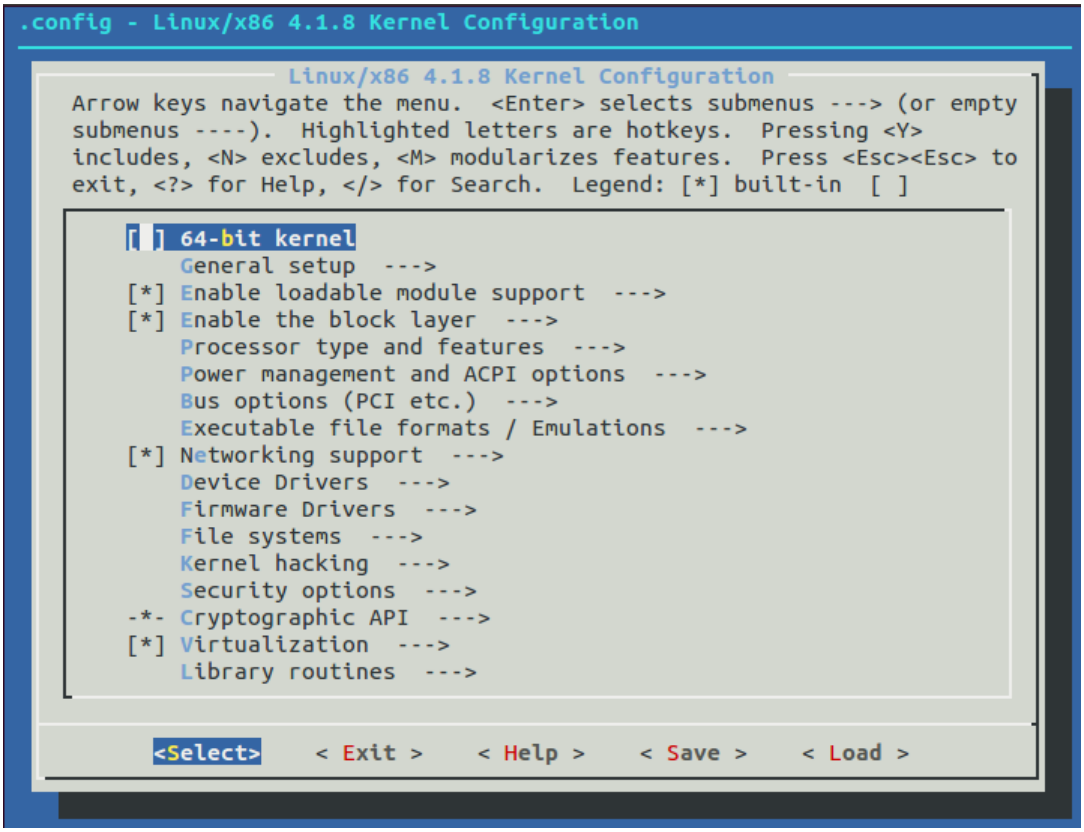



Obr. 5.3: Zavaděč systému U-Boot.




Obr. 5.4: Spuštěný systém Linux

Konfigurace kernelu není jednoduchá a je potřeba se ujistit, že každá provedená změna nezpůsobí problémy při běhu systému. Někteří výrobci hardwaru dávají k dispozici samotnou konfiguraci jádra hardwaru, pro který operační systém sestavujeme, jako v našem případě. Ten již obsahuje veškeré informace týkající se vybraného hardwaru. Pokud ale víme, že je nezbytné zasáhnout do konfigurace, je zde možnost využít dříve uvedený konfigurační nástroj Menuconfig. Ten se postará o to, aby nedošlo k nechtěným chybám způsobeným lidským faktorem. Konfigurační nástroj vygeneruje vše potřebné pro vygenerování výsledného jádra systému. Yocto Projekt je pro tento nástroj již připraven. Pro editaci konfigurace jádra je potřeba sestavit prostředí pomocí nástroje source a BitBake.




Obr. 5.5: Nástroj pro konfiguraci kernelu Menuconfig

Jakmile se spustí konfigurační nástroj, zobrazí se standartní rozhraní [5.5] s možností interaktivně konfigurovat parametry jádra. Po ukončení konfigurace, pro získání výsledných konfiguračních souborů, pouze ukončíme nástroj a potvrdíme uložení změn konfigurace.

### 5.1.6 Úpravy obrazu systému

Konfigurace obrazů systému lze najít v dříve staženém repozitáři. Nachází se v tomto umístění:

```
./sources/meta*
```

Při spouštění sestavovacího procesu jsme jeden z těchto konfiguračních souboru použili. Byl to core-image-minimal. Konfiguraci obrazu rozeznáme podle názvu, kdy je v názvu použit názvem image. Jak bylo dříve uvedeno, konfigurační soubor obsahuje popis obsahu generovaného obrazu systému. Tento konfigurační soubor lze editovat a upravit podle vlastních potřeb.

Zadáním příkazu bitbake -s z umístění projektu se vypíše seznam použitelných balíčků pro vybrané zařízení. Jestliže hledáme konkrétní balíček, lze použít příkaz grep pro odfiltrování z celého seznamu, například takto :

```
$ bitbake -s | grep <komponenta>
```

Některé balíčky mohou být závislé na jiných. Tyto závislosti lze získat například atributem g:

```
$ bitbake -g gnome-desktop3
```

Pro přidání balíčku pro náš operační systém jej musíme přidat do seznamu instalovaných komponent. To provedeme v konfiguračním souboru konkrétního systému, který budeme generovat.
Do proměnné IMAGE_INSTALL přidáme mezerou oddělené vybrané balíčky:

```
IMAGE_INSTALL += "packagegroup-core-c11-sato-games \
                  gnome-desktop3"
```

Pokud chceme přidat balíček k jakémukoliv generovanému systému, do souboru conf/local.conf přidáme řetězec s názvem přidávaného balíku :

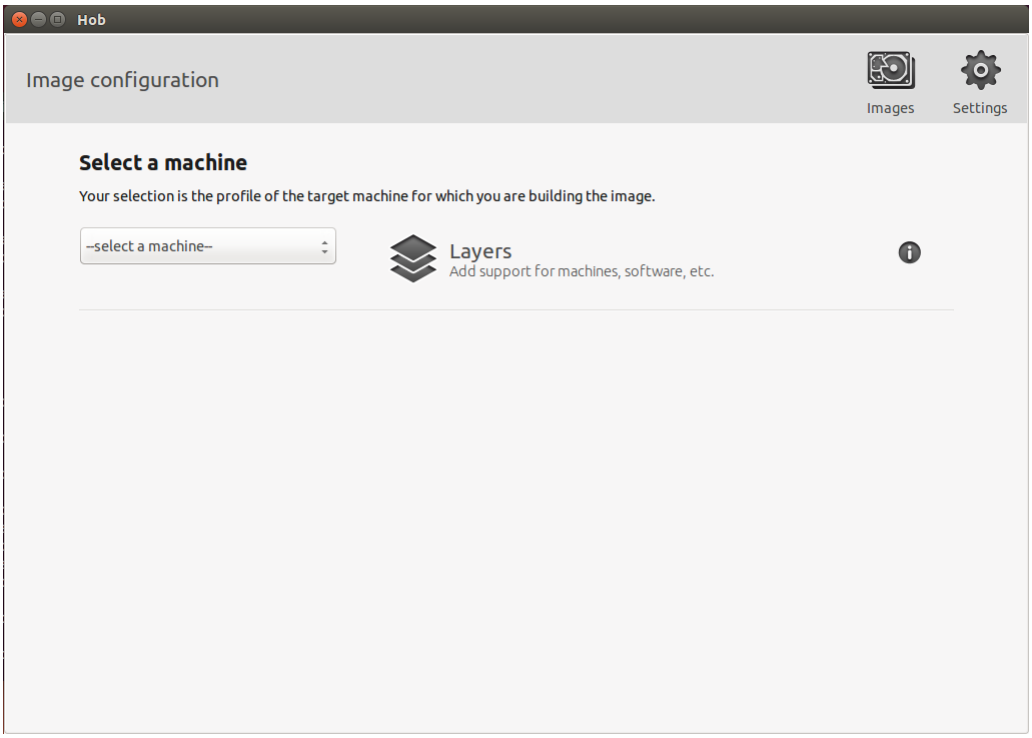
```
CORE_IMAGE_EXTRA_INSTALL += "gnome-desktop3"
```

Poté znovu spustíme přegenerování obrazu systému :

```
$ bitbake core-image-minimal
```

### 5.1.7 Interaktivní nástroj HOB

Hob je grafické uživatelské rozhraní pro BitBake. Hlavním účelem je umožnit uživateli provádět běžné úkony snadněji. S nástrojem Hob lze generovat obrazy systému, upravovat již existující sestavy, vytvoření vlastní sestavy, spouštět vygenerovaný systém v emulátoru QEMU a nasadit vytvořený systém na USB disk nebo SD kartu, pro možnost spuštění na cílovém zařízení.




Obr. 5.6: Úvodní obrazovka grafického rozhraní Hob

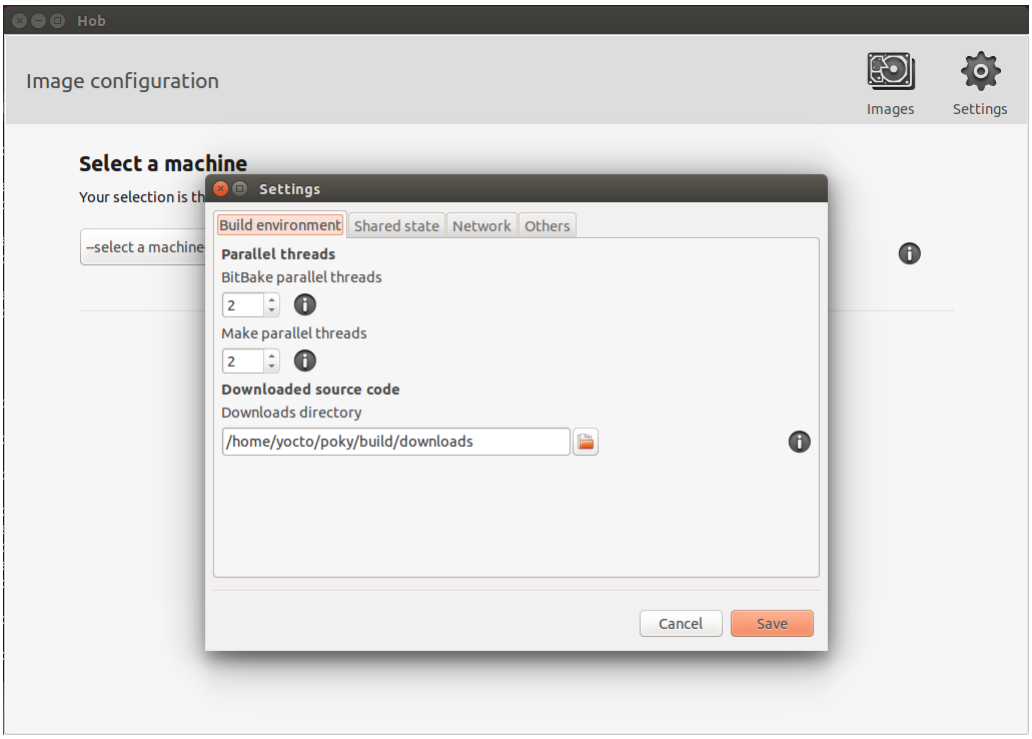
Před vydáním projektu Yocto verze 1.4 měl nástroj Hob vlastní konfigurační soubory, které nejsou nijak propojeny s upravovaným local.conf konfiguračním souborem. Proto oba způsoby generování systému měli různé výstupy.

Od verze 1.4 již byly tyto konfigurační soubory synchronizovány a vzájemně si upravují vybrané proměnné.

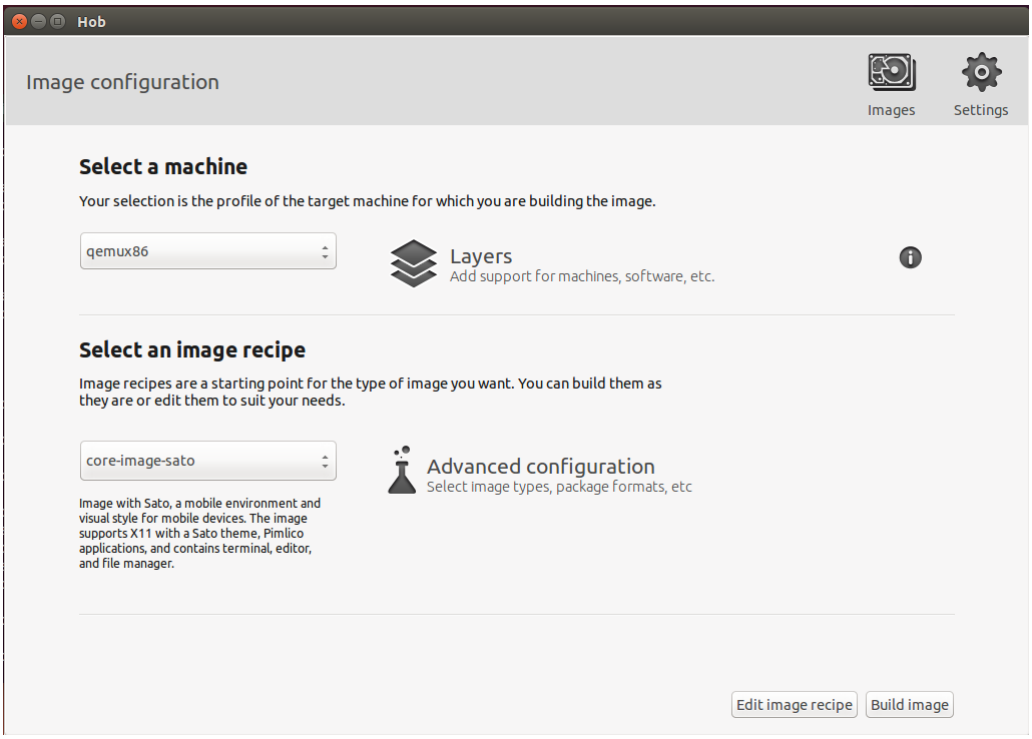
Pro spuštění nástroje Hob je potřeba doinstalovat potřebné balíčky Gtk+ 2.20.0 a PyGtk 2.21.0. Spuštění provedeme zadáním těchto příkazů:

```
$ cd ~/yocto
$ source oe-init-build-env
$ hob
```

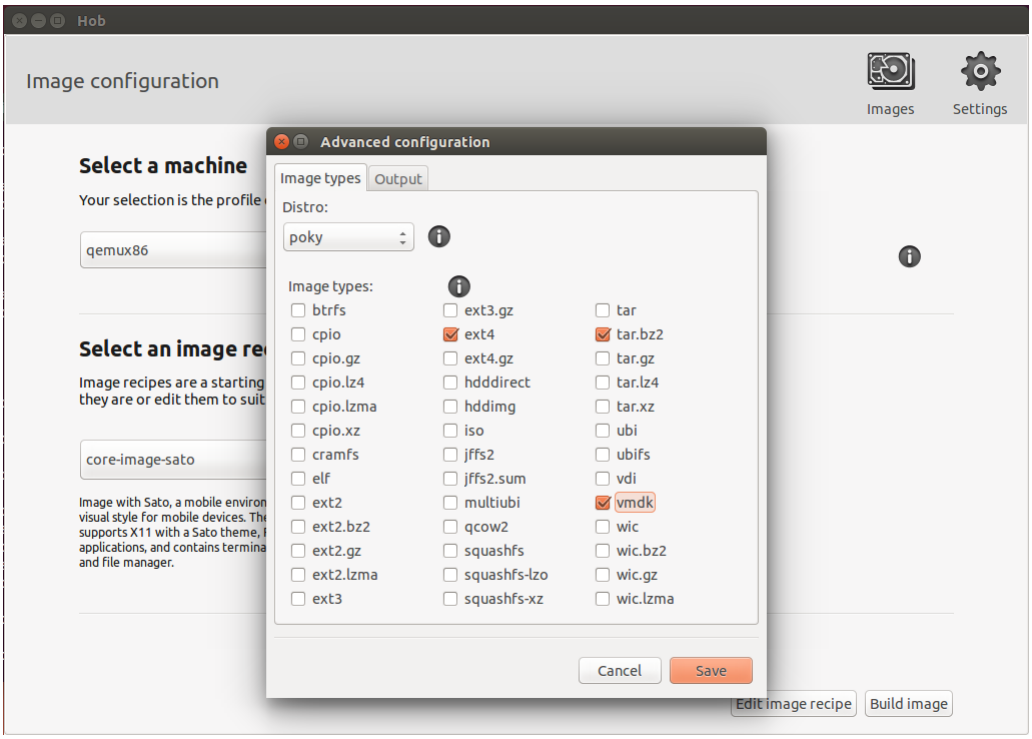
Po spuštění nástroje (5.6) je vhodné ověřit správné nastavení podle toho, jak chceme aby nástroj využíval prostředky systému. Nastavené provedeme kliknutím na ikonu Settings viz. 5.7. Dále si vybereme cílové zařízení na kterém bude systém nasazen a konfiguraci balíčků který má systém obsahovat. V tomto případě byl zvolen qemux86 a core-image-sato viz. 5.8. Po výběru požadovaného systému lze nastavit typy výstupních obrazů. To provedeme po kliknutí na ikonu s názvem Advanced configuration viz. 5.9. Mimo to, je zde možnost nastavení výstupu, tedy formát výstupního balíčku, velikost obrazu, případně přidání vývojového kitu pro



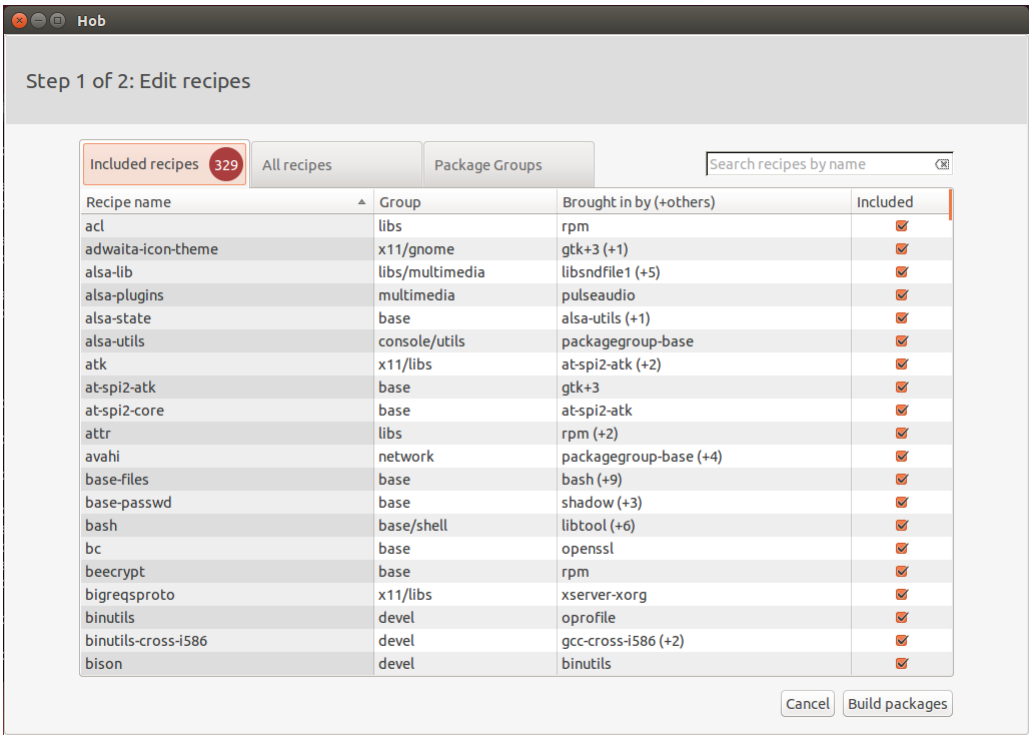

Obr. 5.7: Nastavení prostředků systému pro nástroj Hob



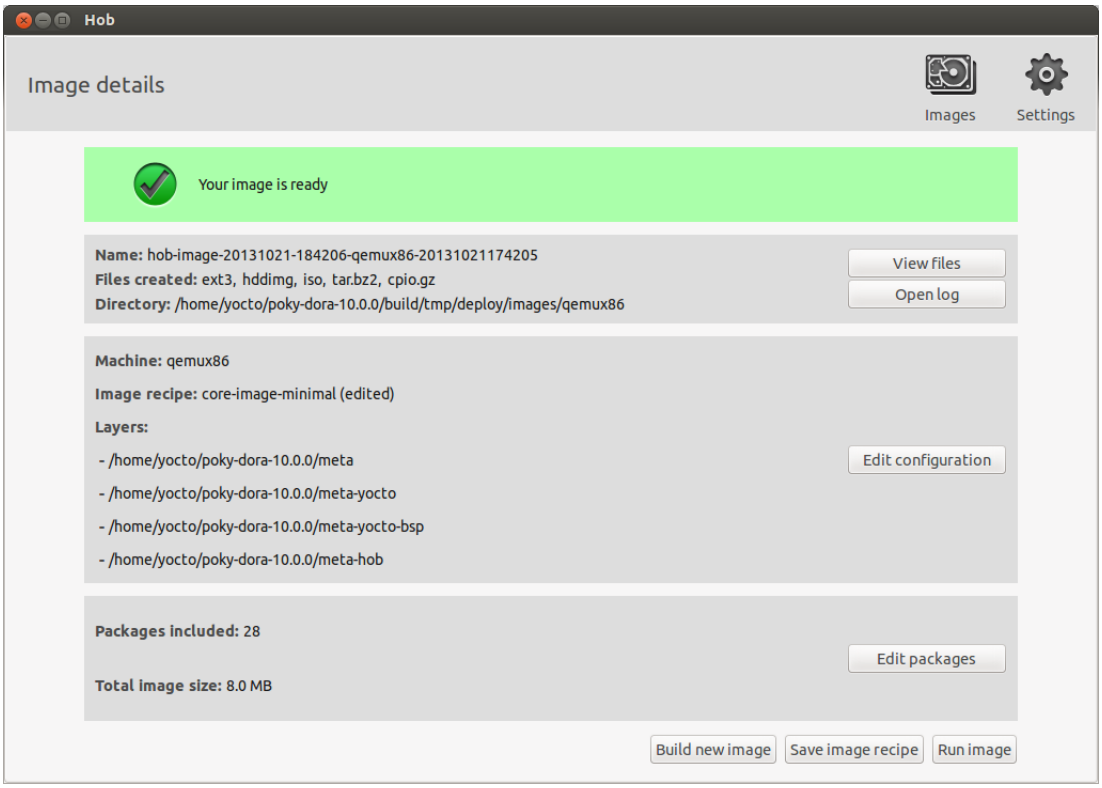

Obr. 5.8: Výběr cílového zařízení a konfigurace obrazu OS




Obr. 5.9: Nastavení typů výstupního obrazu systému




Obr. 5.10: Konfigurace balíčků pro generovaný OS




Obr. 5.11: Výsledek generování obrazu systému

vybranou platformu. Pro dodatečnou editaci obsahu systému je zde možnost přidání nebo odebrání vybraných balíčků, které mají být zahrnuty do systému viz. 5.10. Poté, co je sestava systému kompletní, spustíme proces sestavování kliknutím na tlačítko Build packages. Po sestavení je zde ještě jedna možnost dodatečného přidání nebo odebrání balíčků ze systému. Pro vytvoření obrazu systému klikneme na tlačítko Build image. Po skončení procesu generování obrazu máme v případě cílového zařízení qemux86 možnost vytvořený systém vyzkoušet tlačítkem Run image viz. 5.11. Jestliže je vše v pořádku, můžeme konfiguraci systému aplikovat na konkrétní zařízení.

### 5.1.8 Uživatelské vrstvy

Mimo základní sestavu doplňků, je zde možnost stažení dodatečných balíčků a sestav. Například OpenEmbedded nabízí několik doplňkových vrstev od podpory hardwarů po různé nástroje a grafická rozhraní systému, jako například XFCE nebo Enlightenment. Tento balíček vrstev získáme stažením z repozitáře git do lokálního repozitáře yocto:

```
$ git clone -b jethro git://git.openembedded.org/\
```

meta−openembedded

Přidání vrstev provedeme úpravou proměnné v souboru conf/bblayer.conf například takto :

```
BBLAYERS ?= \
" ${TOPDIR}/src/...
  ...
  ${TOPDIR}/src/meta-openembedded/meta-efl \
  ...
"
```

Nyní můžeme vybírat podporované balíčky i z těchto vrstev. Tedy pokud chceme například přidat grafické rozhraní Enlightenment, upravíme soubor konfigurace obrazu systému, který chceme použít, například core-image-minimal.bb, kde přidáme konkrétní balíky :

```
#add enlightenment window manager
IMAGE_INSTALL += "e-wm"
IMAGE_INSTALL += "terminology"
```

Poté spustíme generování systému opět příkazem bitbake s upraveným konfiguračním souborem:

```
$ bitbake core-image-minimal
```

Po skončení můžeme vygenerovaný systém vyzkoušet a případně doplnit do generátoru chybějící součásti.

Podobným postupem lze postupovat i s nástrojem Hob, tedy stáhnout repozitář openembedded a v grafickém rozhraní viz 5.6 přidat vrstvy kliknutím na tlačítko Layers. V seznamu všech balíčků se již objeví i podpoprované balíky přidaných vrstev. Přidáme tedy potřebné balíčky a dále pokračujeme podle předchozího bodu kapitoly.

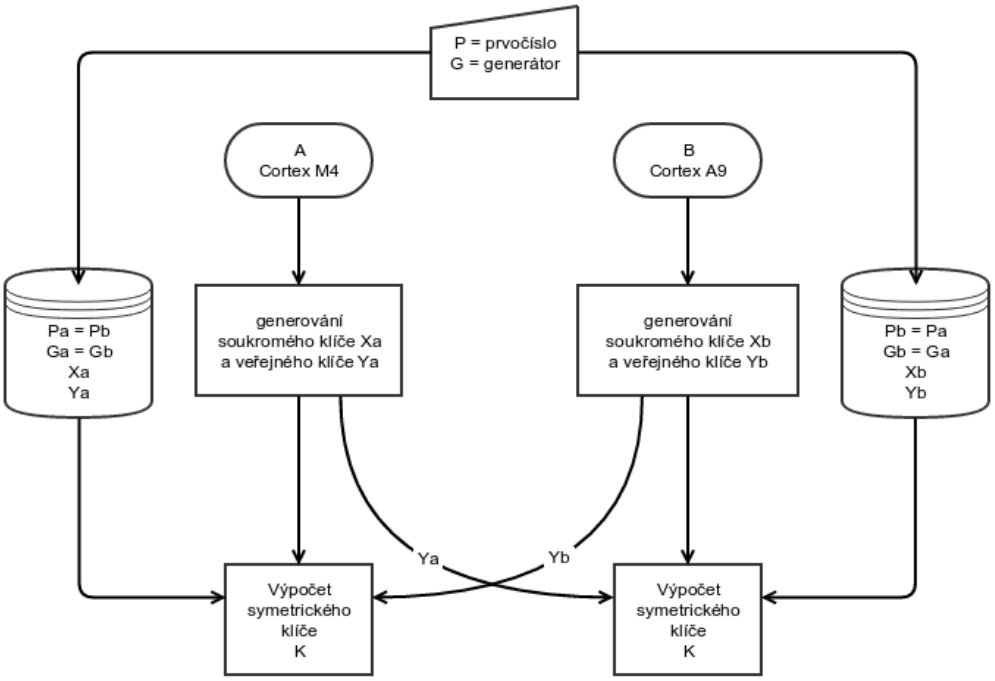
## 5.2 DS-5

Aplikace pro jádro Cortex M4 je potřeba vyvíjet přímo pro tuto platformu, stejně jako sestavení linuxu. Pro kompilaci linuxu byly použity prostředky projektu Yocto. Pro kompilaci aplikace pro jádro M4 bylo použito vývojové prostředí společnosti ARM DS-5 Developmnet Studio. Díky podpoře výrobce jádra tímto softwarem je zajištěna podpora pro kompilaci naší aplikace. Aplikace je vyvíjena pro operační systém reálného času, konkrétně FreeRTOS.

FreeRTOS přináší menší riziko chyb při vývoji aplikací díky vlastnímu aplikačnímu rozhraní. Konfiguraci systému FreeRTOS pro vybraný hardware i.MX 6SoloX

lze sehnat přímo u výrobce mikrokontroléru s dokumentací dostupných funkcí aplikačního rozhraní.

Pro demonstraci zabezpečené komunikace mezi jádry bylo nutné generovat data, která mají být bezpečně přenášena do linuxové aplikace. K tomu poslouží akcelerometr obsažený na vývojové desce, připojený k mikrokontroléru sběrnicí i2c. Při vývoji jsem vycházel z ukázkových aplikací dodávanými v softwarovém balíčku výrobce, který obsahoval aplikace pro akcelerometr i RPMSG. Výstupem aplikace pro akcelerometr jsou hodnoty aktuálních pozic na osách akcelerometru. Aplikace pro RPMSG obsahuje mimo jiné funkce pro odesílání a příjem zpráv, které jsou potřeba pro tento projekt. Nejprve v aplikaci probíhá inicializace hardwaru jádra a




Obr. 5.12: Blokové schéma algoritmu Diffie-Hellman.

dostupných prostředků, inicializace sběrnice i2c pro akcelerometr, inicializace virtuální sběrnice RPMSG a alokace paměti pro přenášená data. Poté probíha proces výměny klíčů.

Jak již bylo řečeno, jako algoritmus výměny klíčů bude použit Diffie-Hellman. Princip algoritmu je založen na asymetrické kryptografii s veřejným a soukromým klíčem. Využívá se praktické početní operace modulo, neboli zbytek po dělení. Proto je nutné v těchto algoritmech použití prvočísel, aby modulo nebylo nulové. Teorie je taková, že při dělení dvou čísel, po kterém zůstane zbytek, tento zbytek o původních

dvou číslech nic neřekne a je velice náročné odhadnout pouze ze zbytku o jaká čísla šlo.

Praktický postup procesu získání symetrického klíče je znázorněn na obr.[5.12]. Dopředu jsou stanoveny čísla P a G, což jsou prvočíslo a generátor. Tato čísla jsou veřejná a jsou natvrdo zapsána v aplikacích pro obě jádra. Dalším krokem je vygenerování soukromého čísla pomocí generátoru náhodných čísel. Je důležité zajistit jedinečnost a neopakovatelnost tohoto čísla. Následuje výpočet veřejné části klíče. Ta se počítá z generátoru, soukromého čísla a prvočísla podle vzorce[5.1]. Provede se výměna veřejných klíčů a vypočítá se symetrický klíč podle vzorce[5.2]. Tento klíč může sloužit pro další šifrovanou komunikaci, nebo se tímto klíčem přenese nový náhodně vygenerovaný klíč.

$$Y_{\mathrm{a}} = G^{X_{\mathrm{a}}} (\mathrm{mod} P) \ldots Y_{\mathrm{b}} = G^{X_{\mathrm{b}}} (\mathrm{mod} P) \tag{5.1}$$

$$K = Y_{\mathrm{b}}^{X_{\mathrm{a}}} (\mathrm{mod} P) \ldots K = Y_{\mathrm{a}}^{X_{\mathrm{b}}} (\mathrm{mod} P) \tag{5.2}$$

Hlavní proces programu je vypsán v souboru main.h. Na začátku souboru jsou vypsány pomocné soubory, které mají být zahrnuty do programu. Jako další je nadefinování proměnných. Například nadefinování prvočísla a generátoru provedeme takto:

```
const uint8_t p[] = {
    0x00,0x8d,0x08,0x2d,0xf8,0x00,0x78,0xc7,0x34,0xaa,
    0xe6,0x47,0xc7,0xbe,0xc1,0x22,0x84,0xc7,0xfb,0x70,
    0xcf,0xd4,0x4b,0x98,0x50,0x97,0x6b,0x6e,0x16,0xeb,
    0x99,0x04,0x11,0x83,0x43,0x0b,0x21,0x8d,0xd1,0xe0,
    0x46,0xc8,0x04,0x4f,0x89,0x52,0xc4,0x74,0x5c,0x83,
    0xb8,0x36,0x3e,0x3f,0x8b,0x39,0xf3,0x00,0xd1,0x7c,
    0x52,0x2a,0xbc,0x63,0x37,0xcd,0x6f,0xcf,0x4c,0x65,
    0x20,0xfd,0x0a,0xcc,0xee,0x72,0x97,0x5c,0x72,0x28,
    0xa5,0xbe,0x11,0x15,0x5a,0x95,0x4f,0xb2,0x0c,0x5d,
    0xd6,0x69,0x70,0x6e,0x2a,0x1a,0x2e,0x25,0x94,0x12,
    0xae,0x01,0x9c,0x1a,0x94,0x74,0x8a,0x37,0x2f,0x91,
    0x3c,0x6c,0x3c,0x17,0x58,0x8b,0x29,0xcf,0xa2,0xf7,
    0x8d,0x96,0xf4,0x57,0xff,0x71,0xe0,0xfa,0x3b};
const uint8_t g[]={0x02};
```

Dále si nadefinujeme globální proměnné pro ukládání a práci s klíči a zásobníky pro tyto klíče a zprávy odesílané přes RPMSG. Ve funkci main probíhá veškerá inicializace používaného hardwaru a vytváří se potřebné úkoly pro FreeRTOS jako obsluha akcelerometru a RPMSG.

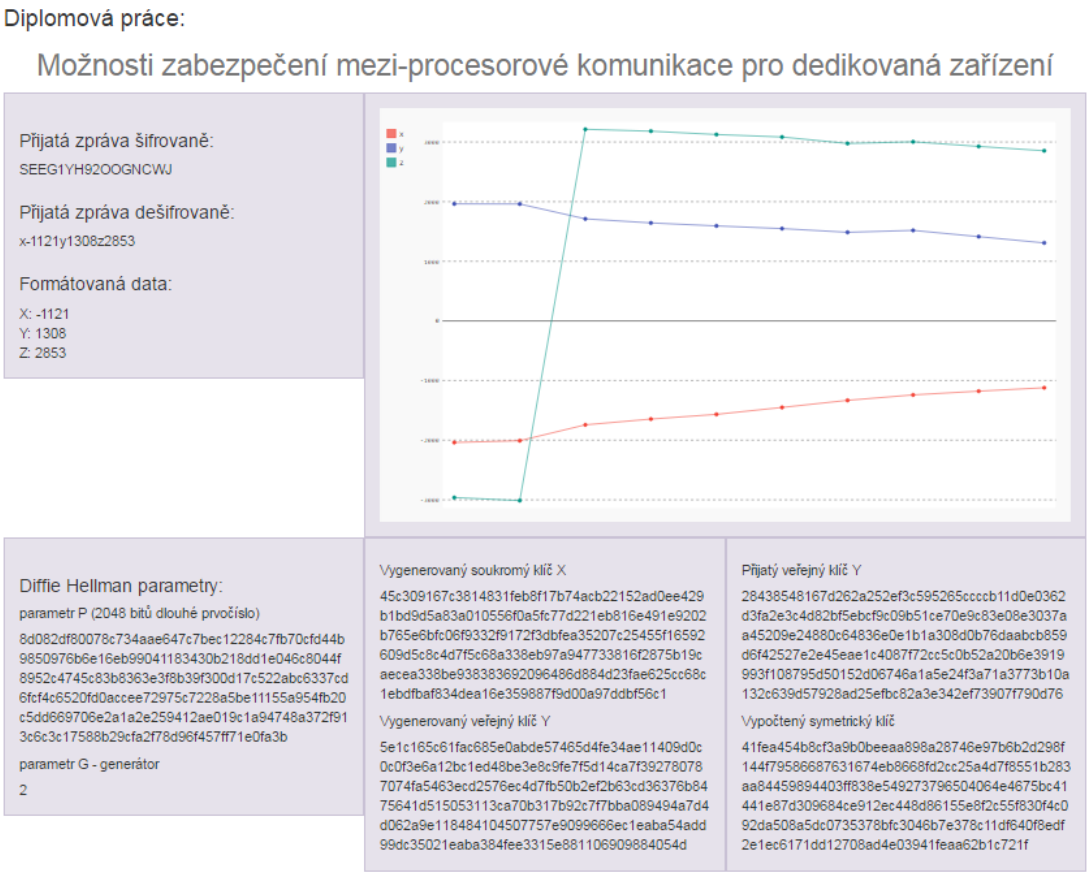
```
int main(void)
{
    hardware_init();
    PRINTF("hardware init");

    MU_Init(BOARD_MU_BASE_ADDR);
    NVIC_SetPriority(BOARD_MU_IRQ_NUM, APP_MU_IRQ_PRIORITY);
    NVIC_EnableIRQ(BOARD_MU_IRQ_NUM);

    /* Create a task. */
    xTaskCreate(StrEchoTask,
                "String Echo Task",
                APP_TASK_STACK_SIZE,
                NULL,
                tskIDLE_PRIORITY+1,
                NULL);

    xTaskCreate(accelerometerTask,
                "Accelerometer",
                APP_TASK_STACK_SIZE,
                NULL,
                tskIDLE_PRIORITY+1,
                NULL);

    /* Start FreeRTOS scheduler. */
    vTaskStartScheduler();

    /* Should never reach this point. */
    while (true);
}
```

Proces pro akcelerometr ve smyčce čte hodnoty z akcelerometru a zapisuje do proměnné. Funkce pro RPMSG začíná algoritmem pro předání klíče. Nejprve čeká dokud nedostane žádost o veřejný klíč, ten poté společně se soukromým vygeneruje a veřejnou část odešle druhému jádru. To v zápětí odešle vlastní vygenerovaný veřejný klíč a obě jádra si ze získaných hodnot spočítají sdílený soukromý klíč.

Následná komunikace je šifrována algoritmem pro symetrickou kryptografii AES (Advanced Encryption Standard) a po 16 bajtových blocích je odesílána druhému

jádru zašifrovaná 128 bitovým klíčem.

## 5.3 Demonstrace




Obr. 5.13: Vytvořená webová stránka.

Pro demonstraci zabezpečené mezi-jádrové komunikace byla v Linuxu vytvořena aplikace v programovacím jazyce Python. Pomocí modulů přidaných do konfiguračního souboru Yocta pro generování Linuxu byly doinstalovány potřebné moduly jako Flask pro webový server, pyserial pro komunikaci s druhým jádrem a další pomocné balíčky. V Pythonu byl vytvořen webový server s aplikací prezentující proces komunikace[5.13]. Aplikace se v půl sekundovém intervalu dotazuje druhého jádra o zaslání šifrovaných dat z akcelerometru, provede dešifrování a výsledné hodnoty zakreslí do tabulky. Pro ukázku jsou zde zobrazeny i jednotlivé parametry postupu předání klíče a šifrování. Tyto parametry však v praxi nesmí být veřejnosti přístupné.

# 6 ZÁVĚR

Zadáním diplomové práce bylo seznámení se s mikrokontroléry rodiny i.MX společnosti NXP. V diplomové práci byly popsány verze mikrokontrolérů i.MX, seznámili jsme se s dostupnými periferiemi a prostředky. Z daných mikrokontrolérů byl vybrán i.MX 6SoloX, vybavený dvěma procesorovými jádry Cortex A9 a Cortex M4, který umožňuje mezi-procesorovou komunikaci.

Dále byly probrány způsoby mezi-procesorové komunikace jako MCC a RPMSG. Po předchozí studii jednotlivých možností komunikace byl vybrán protokol RPMSG.

Bylo zadáno, aby byl mikrokontrolér podpořen build systémem Yocto. V teoretickém úvodu byly nejprve popsány možnosti implementace operačních systému pro dedikovaná zařízení s historií vývoje. Build systém Yocto nabízí usnadnění portace linuxové distribuce pro dedikovaná zařízení. Díky podpoře výrobce hardwaru konfiguračními soubory pro projekt Yocto, byl vygenerován operační systém Linux, který tak mohl být nasazen na výkonnější jádro mikrokontroléru.

V diplomové práci jsme se věnovali možnostem zabezpečení mezi-jádrové komunikace, které vybraný hardware nabízí. Bylo navrženo řešení postavené na hardwarových akcelerátorech, ale z důvodu nedostatku dostupných informací potřebných pro realizaci, byla zvolena cesta softwarovým řešením. Byl proto navržen systém zabezpečení pomocí softwarové knihovny podporující zabezpečenou výměnu klíčů, algoritmus Diffie-Hellman a šifrovací algoritmus AES pro následnou zabezpečenou komunikaci mezi jádry.

Pro demonstraci řešení byl použit akcelerometr připojený k mikrokontroléru pomocí sběrnice i2c. Ten je ovládán slabším jádrem Cortex M4, pro který byl napsán obslužný program postavený na systému reálného času FreeRTOS. V tomto programu byla rovněž implementována knihovna CycloneSSL, starající se o zabezpečení přenosu dat získaných z akcelerometru. Na druhém jádře byla vytvořena Python aplikace pro vygenerovanou distribuci Linuxu. Aplikace spouští naprogramovaný webový server, který prezentuje zabezpečenou komunikaci vyobrazením parametrů výměny klíčů, přijetím šifrovaných dat a příklad zobrazení dat pomocí grafu vykreslující poslední přijaté dešifrované hodnoty.

Prezentovaným výsledkem této práce je funkční demonstrační aplikace se zabezpečeným přenosem dat mezi dvěma procesorovými jádry.

# LITERATURA


[1] LOOMIS, D. *The History of Operating Systems* [online]. 2013 [cit. 15. 12. 2015]. Dostupné z URL: <http://internationalprogrammersday.org/history-of-operating-systems>.

[2] OMAPpedia *Design Overview - RPMsg* 10. 11. 2011 [cit. 20. 5 2016]. Dostupné z URL: <http://omappedia.org/wiki/Design_Overview_-_RPMsg>.

[3] Wikipedia *i.MX* 18. 5. 2016 [cit. 20. 5 2016]. Dostupné z URL: <https://en.wikipedia.org/wiki/I.MX>.

[4] Freescale *i.MX 6SoloX Reference Manual* 17. 2. 2015 [cit. 20. 5 2016]. Dostupné z URL: <http://cache.nxp.com/files/32bit/doc/ref_manual/IMX6SXRM.pdf?fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation&fileExt=.pdf>.

[5] Freescale *i.MX Sabre SDB* 12. 2. 2016 [cit. 20. 5 2016]. Dostupné z URL: <http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx6qp/sabre-board-for-smart-devices-reference-design-based-on-the-i.mx-6-series:RDIMX6SABREBRD?fpsp=1&tab=Documentation_Tab#nogo>.

[6] Wolfe, D. *Multi-Core Communication* 6. 2015 [cit. 20. 5 2016]. Dostupné z URL: <https://community.freescale.com/servlet/JiveServlet/downloadBody/105770-102-1-27001/FTF-DES-F1133.pdf>.

[7] WALLS, C. *Selecting an operating system for an embedded application* [online]. 25. 10. 2014 [cit. 15. 12. 2015]. Dostupné z URL: <http://www.embedded.com/design/operating-systems/4436454/Selecting-an-operating-system-for-an-embedded-application->.

[8] KRZYZANOWSKI, P. *Operating Systems* [online]. 2015 [cit. 15. 12. 2015]. Dostupné z URL: <https://www.cs.rutgers.edu/~pxk/416/notes/01-intro.html>.

[9] RIFENBARK, S. *Yocto Project Development Manual* [online]. 2015 [cit. 15. 12. 2015]. Dostupné z URL: <http://www.yoctoproject.org/docs/2.0/dev-manual/dev-manual.html>.

[10] BAKER, M., ROZEMA, G. *OpenWrt* [online]. poslední aktualizace 2015 [cit. 15. 12. 2015]. Dostupné z URL: <https://openwrt.org/>.

[11] BARRY, R. *FreeRTOS* [online]. poslední aktualizace 2015 [cit. 15. 12. 2015]. Dostupné z URL: <http://www.freertos.org/>.

# SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

| | |
|---|---|
| AES | Pokročilý standart šifrování – Advanced Encryption Standard |
| AMP | Asymetrická multiprocesorová konfigurace – Asymetric Multiprocessing |
| BIOS | Základní vstupně-výstupní funkce – Basic Input-Output System |
| BSD | Berkeley Software Distribution |
| CAAM | Akcelerovaný kryptografický a zabezpečovací modul – Cryptographic Acceleration and Assurance Module |
| CAN | Síťový protokol – Controller Area Network |
| CP/M | Řídíci program pro mikropočítače – Control Program for Microcomputers |
| CPU | centrální procesorová jednotka – central processing unit |
| CSU | Centrální zabezpečovací jednotka – Central Security Unit |
| DDR | Double Data Rate |
| DHCP | Dynamický konfigurační protokol hosta – Dynamic Host Configuration Protocol |
| DNS | Systém doménových názvů – Domain Name System |
| DOS | Diskový operační systém – Disk Operating System |
| FTP | Protokol přenosu souborů – File Transport Protocol |
| FullHD | Obraz s plným 1080p rozlišením – Full High Definition |
| GUI | Grafické uživatelské rozhraní – Graphical User Interface |
| HAB | Systém zavaděče s vysokým zabezpečením – High Assurance Boot |
| HDMI | Multimediální rozhraní s vysokým rozlišením – High-Definition Multi-media Interface |
| HTTP | Protokol hypertextového přenosu – Hypertext Transfer Protocol |
| HW | Hardware |
| IP | Internetový protokol – Internet Protocol |

| | |
|---|---|
| JTAG | Joint Test Action Group |
| JVM | Java virtuální stroj – Java Virtual Machine |
| LCD | Displej s tekutými krystaly – Liquid Crystal Display |
| LVDS | Nízkonapěťové diferenční signály – Low-voltage Differential Signaling |
| MCC | Multi-jádrová komunikace – Multi Core Communication |
| MCP | Master Control Program |
| NIST | Národní institut standardů a technologie – National Institute of Standards and Technology |
| OCOTP | Jednou programovatelná paměť na čipu – On-Chip One Time Programmable Memory |
| OCRAM | Zabudovaná paměť s náhodným přístupem v čipu – On-Chip Random Memory Access |
| OMAP | Otevřená platforma pro multimediální aplikace – Open Multimedia Applications Platform |
| OTG | USB On-The-Go |
| PCIe | Systémová sběrnice – Peripheral Component Interconnect Express |
| PPP | Protokol síťové vrstvy – Point-to-Point Protocol |
| QSPI | Sériová sběrnice periferií s frontou – Queued Serial Peripheral Interface |
| RAM | Paměť s náhodným přístupem – Random Access Memory |
| RDC | Řadič doménových zdrojů – Resource Domain Controller |
| RPMSG | Komunikace mezi procesory – Remote Processor MEssaging |
| RSA | Šifra s veřejným klíčem – Rivest, Shamir, Adleman |
| RTOS | operační systém reálného času – real-time operating system |
| S-ATA | Sériová sběrnice připojování pokročilé technologie – Serial Advanced Technology Attachment |
| SD | Paměťová karta – Secure Digital |

| | |
|---|---|
| SDIO | Rozhraní s bezpečným digitálním vstupem/výstupem – Secure Digital Input Output |
| SDMA | Mezerou oddělený několikanásobný přístup – Space-division multiple access |
| SJC | Systémový ovladač JTAG – System JTAG Controller |
| SMP | Symetrická mmulti-procesorová konfigurace – Symetric Multiprocessing |
| SNMP | Protokol s jednoduchou síťovou správou – Simple Network Management Protocol |
| SNVS | Zabezpečená energeticky nezávislá paměť – Secure Non Volatile Storage |
| SW | Software |
| SoC | Jednočipový systém – System-on-Chip |
| TCP | Protokol s řízeným přenosem – Transport Control Protocol |
| TZ | Bezpečná zóna – TrustZone |
| USB | Univerzální sériová sběrnice – Universal Serial Bus |