VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PROBLEMATIKA PŘECHODU OD JEDNOJÁDROVÉ K VÍCEJÁDROVÉ IMPLEMENTACI OPERAČNÍHO SYSTÉMU

ISSUE OF MIGRATING FROM SINGLE-CORE TO MULTI-CORE IMPLEMENTATION OF OPERATING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAKUB SKOPAL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce se zabývá úpravami hardwarového návrhu a operačních systémů vícejádrové platformy ZedBoard tak, aby bylo možné využít obě jádra procesoru ARM Cortex A9 obsažená v SoC Zynq7000. Rozebírá obecnou problematiku vícejádrového prostředí a základní funkce jádra a operačního systému. Popisuje zvolené realizační prostředky ZedBoard a FreeRTOS. V realizační části jsou demonstrovány konkrétní kroky při převodu jednojádrového operačního systému na vícejádrový, ale také kroky nutné ke spuštění dvou různých operačních systémů na dvou jádrech. V poslední časti jsou shrnuty všechny dosažené výsledky.

Abstract

This thesis deals with the modifications of the hardware design and operating systems of the ZedBoard multi-core platform so that both ARM Cortex A9 processor cores included in SoC Zynq7000 can be used. It analyses the general issue of the multi-core environment and the core functions of the kernel and the operating system. It describes selected means of implementation ZedBoard and FreeRTOS. In the implementation section, specific steps are demonstrated to convert a single-core operating system to a multi-core system but also steps required to run two different operating systems on two processor cores. In the last section all achieved results are summarized.

Klíčová slova

FreeRTOS, ZedBoard, Cortex-A9, vícejádrová platforma, vícejádrové prostředí, vícejádrový procesor, operační systém, Linux, AMP, SMP

Keywords

FreeRTOS, ZedBoard, Cortex-A9, multicore platform, multicore environment, multicore procesor, operating system, Linux, AMP, SMP

Citace

SKOPAL, Jakub. *Problematika přechodu od jednojádrové k vícejádrové implementaci operačního systému*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Strnadel Josef.

Problematika přechodu od jednojádrové k vícejádrové implementaci operačního systému

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana inženýra Josefa Strnadela, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Skopal
23. května 2017

Poděkování

Děkuji panu Ing. Josefovi Strnadelovi, Ph.D., za odborné vedení, trpělivost, vstřícnost při řešení této práce a zapůjčení dvou realizačních prostředků ZedBoard.

Obsah

Kapitola 1

Úvod

Počítače a vestavěná zařízení jsou dnes umisťovány téměř do všech elektronických zařízení. Tyto systémy se nachází v běžné domácnosti (pračky, mikrovlnky, televize atd.), ale i na každém kroku venku (v autě, mobilní telefon atd). Tato zařízení jsou navržena tak, aby usnadnila svým uživatelům interakci s prostředkem, který využívají. Současná doba, ve které žijeme, se neustále zrychluje, a proto je třeba zrychlovat i tato zařízení.

Zrychlování těchto zařízení bylo dříve prováděno zvyšováním frekvence jednojádrových procesorů a vylepšováním implementace emulace paralelismu, například technologie Hyper-Threading od firmy Microsoft. Tento přístup však narazil na fyzikální limity, které nelze snadno řešit. Problémem je zejména regulování energetických a tepelných ztrát při vysokých frekvencích. Z tohoto důvodu je nutné změnit dosavadní principy zvyšování výkonu, a to přidáním více výpočetních jednotek do procesoru, které by mohly pracovat souběžně a nezávisle na sobě. Vznikl tedy nápad vícejádrových procesorů, jež často fungují efektivněji i při nižších frekvencích. Teplotní a energetické ztráty jsou nižší.

Výhodou vícejádrových systémů je možnost zpracovávat více instrukcí souběžně. S tím úzce souvisí operační systém, který se stará o pokud možno co nejrovnoměrnější rozdělení úloh jednotlivým jádrům procesoru. V operačních systémech, jako je například Windows, Linux, Android atd., jsou již vícejádrové systémy podchyceny a plánovače těchto operačních systémů již podporují práci ve vícejádrovém prostředí. To ale neplatí pro vestavěná zařízení a Real-Time operační systémy, které většinou pracují pouze na jednom jádře. Jelikož ale současná doba požaduje vyšší výkon ve všech oblastech, je nutné i v Real-Time operačních systémech tyto požadavky reflektovat. Převod těchto operačních systémů jistě není jednoduchý a je třeba porozumět základním pojmům a principům, které se při základních úkonech provádějí (plánování, správa paměti, synchronizace atd.).

1.1 Cíle práce

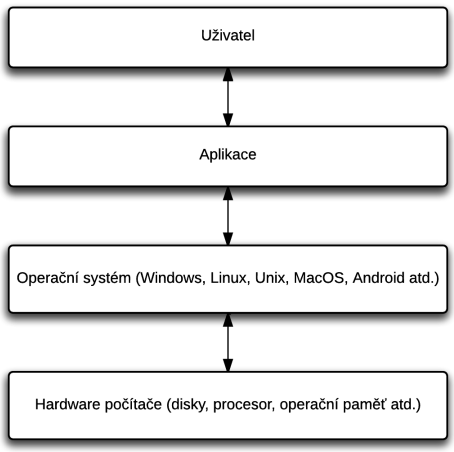
Primárním cílem mé práce je rozebrání problematiky převodu jednojádrového operačního systému na vícejádrový. Jako jeden z dalších cílů této diplomové práce jsem zařadil i realizaci SMP a AMP konfigurací na zvoleném realizačním prostředku ZedBoard. Pro dosažení těchto cílů se zabývám rozborem základních funkcí operačních systémů a principy těchto funkcí ve vícejádrovém prostředí. Po pochopení zkoumané problematiky jsem navrhl nutné změny pro transformaci jednojádrového operačního systému na vícejádrový a otestoval SMP i AMP konfiguraci.

1.2 Oblasti a pojmy související s tématem práce

1.2.1 Operační systém

Počítačový systém se obvykle skládá z jednoho nebo více procesorů, operační paměti, úložišť (pevné disky, flash disky, eMMC disky), tiskárny, klávesnice, zobrazovacího zařízení (LCD displej, monitor), sítového rozhraní a různých vstupně-výstupních zařízení. Všechny tyto součásti tvoří komplexní systém. Počítačový program, který bude pracovat s hardwarem počítače a udržovat informace o všech těchto komponentech, je nazýván operačním systémem [44].

Hlavními úlohami operačního systému jsou správa všech připojených zařízení a poskytnutí jednoduchého rozhraní pro práci se zařízeními počítače. Operační systém vytváří vrstvu mezi hardwarem počítače a uživatelem [44].



Obrázek 1.1: Vrstvy počítačového systému [44].

Operační systém poskytuje uživateli prostředí pro řízení komplexního počítačového systému. Uživatel nemá povědomí o všem, co je nutné vykonat, například při spouštění aplikací. Tento průběh je před uživatelem zcela skrytý, a tím uživateli značně zjednodušuje práci s hardwarem počítače [44].

Základní funkce operačního systému jsou popsány v kapitole 1.3.

Základní klasifikace operačních systémů

Jednoúlohový a víceúlohový

V jednoúlohovém operačním systému může být v jednom okamžiku spuštěna pouze jedna úloha, zatímco ve víceúlohovém operačním systému může být spuštěno více úloh souběžně. Tento souběžný běh úloh je realizován pomocí sdílení procesorového času. Ve skutečnosti je ve víceúlohovém operačním systému spuštěna v jeden okamžik vždy jedna úloha [16].

Sdílení procesorového času zajišťuje plánovač, který rozdělí dle zvoleného plánovacího mechanismu procesorový čas úlohám, které běží na procesoru právě dobu určenou plánovačem. Tato přepínání kontextů úloh jsou velmi rychlá, a vytváří tak dojem souběžnosti více úloh. Plánovač operačního systému je popsán v kapitole 1.3.1. [16].

Jednouživatelský a víceuživatelský

Víceuživatelské operační systémy umožňují souběžnou interakci s více uživateli. Data ve víceuživatelském operačním systému mohou být sdílena mezi uživateli, ale i skrývána před ostatními uživateli. Poskytuje základní ochranu dat jednotlivých uživatelů [16].

Rozšiřuje koncept jednouživatelského operačního systému. Víceuživatelský operační systém dokáže rozlišit prostředky přidělené uživatelům a identifikovat úlohy jednotlivých uživatelů. Operační systém pak zajistí správné rozdělení prostředků a procesorového času mezi jednotlivé uživatele [16].

1.2.2 Paralelní systémy

Paralelní systém se skládá z více procesorů, které jsou pevně propojeny. Tyto procesory mezi sebou sdílí sběrnice, paměť, vstupně-výstupní zařízení, hodiny, periferie atd. Využívá se paralelního zpracování, kdy je jedna úloha zpracovávána více procesory najednou, a tudíž je snížen čas potřebný pro vykonání této úlohy. Paralelní systémy se dnes běžně nazývají superpočítače [16].

Distribuovaný operační systém

Distribuovaný operační systém je kolekce autonomních systémů, které jsou schopny mezi sebou komunikovat a spolupracovat pomocí například ethernetového rozhraní. Distribuovaný systém sdružuje autonomní systémy a vytváří abstraktní virtuální stroj pro uživatele [16].

Distribuovaný systém se jeví pro uživatele jako klasický centralizovaný operační systém, ale ve skutečnosti běží na několika na sobě nezávislých procesorech. Klíčová vlastnost distribuovaných systémů je zapouzdření problémů spolupráce jednotlivých procesorů, uživatel se systémem pracuje jako s klasickým jednoprocesorovým systémem. Ve skutečnosti tyto systémy dokážou spouštět aplikace na více procesorech zároveň, a tím značně zvyšují využití procesorového času aplikací. Distribuované systémy jsou více spolehlivé, dokážou pracovat, i když některý autonomní systém přestal pracovat správně [16].

1.2.3 Vestavěné systémy

Vestavěné systémy neboli embedded systémy, jsou zařízení, která obsahují software a hardware ve velmi těsném spojení. Většinou jsou tyto systémy navrženy k řešení konkrétních úloh tak, aby interakce s uživatelem byla minimální nebo žádná. Systémy tohoto typu interagují s procesy nebo s prostředím. Rozhodnutí, která jsou vytvářena v reálném čase za chodu systému, jsou závislá na vstupních informacích systému. Jsou reaktivní a zpracovávají vstupní informace v reálném čase. Oproti osobním počítačům mají tyto systémy značně omezenou kapacitu paměti a diskového prostoru, výpočetní sílu a také se u těchto systémů předpokládá nízká spotřeba i nízká cena [25].

Vestavěný operační systém

Vestavěný operační systém musí být vytvořen tak, aby mohl být spuštěn na vestavěných zařízeních, jako jsou například chytré telefony, chytré hodinky, mikrovlnky atd. Nejrozšířenějšími vestavěnými operačními systémy jsou Linux, Windows CE, Minix a různé Unix systémy [25].

Real-Time operační systém

Real-Time operační systém je definován jako systém, ve kterém správnost výpočtů nezávisí pouze jen na logické správnosti výpočtu, ale i na času, ve kterém byl výsledek produkován. Lze říci, že Real-Time operační systém musí dodržovat přísná časová omezení. Vstupy, data a výstupy by měly být k dispozici během určité časové periody, jinak může dojít k selhání systému. Například obsluhuje-li Real-Time operační systém airbagy v autě, musí být v případě nehody airbag vystřelen v pěvně definovaném čase, jinak může dojít k vážnému zranění posádky auta. Běžný operační systém by nemohl tak rychle zareagovat, proto se v těchto případech používají Real-Time operační systémy, které dle typu garantují odezvu na nějaký podnět v určitém čase. Real-Time systémy pracují správně pouze tehdy, pokud dokážou vytvořit správnou odezvu na podnět v přesně definovaném časovém úseku [16].

Typy Real-Time operačních systémů

Rozdíly mezi typy Real-Time operačními systémy jsou uvedeny v následující tabulce:

Hard Real-Time operační systém	Soft Real-Time operační systém
Autonomní detekce chyb.	Uživatelem asistovaná detekce chyb.
Limitovaná možnost navrácení/zotavení z chyb vlivem krátkodobé integrity dat.	Dlouhodobá integrita dat, navrácení/zotavení z chyb je jednodušší.
Pracuje s menšími shluky dat.	Pracuje s většími shluky dat.
Ochrana dat je často kritická.	Ochrana dat nemusí být kritická.
Předvídatelné (dobře definované) špičkové zatížení.	Degradovaný výkon při špičkovém zatížení.
Použití: kontrola letištního provozu (odezva musí být rychlá, jinak může dojít k fatálním následkům).	Použití: telekomunikace (přenos hlasu) - Při zpoždění přenosu hlasu nedojde k závažným následkům.

Tabulka 1.1: Porovnání hard Real-Time systému a soft Real-Time systému [16].

Hard Real-Time operační systém garantuje dokončení kritických úloh v přesně definovaném časovém úseku, tj. všechna zpoždění v systému musí být předvídatelná. Zatímco v soft Real-Time operačním systému je kritickým úlohám přiřazena vyšší priorita než běžným úlohám. Zpoždění v systému potřebují být předvídatelná, aby se zaručilo nehladovění kritických úloh [16].

V soft Real-Time systému mohou být mohou být časová omezení občas nedodržena nebo dokonce přeskočena. Také mohou být v různých případech překročena o velmi malý časový úsek, zatímco v hard Real-Time operačním systému musí být všechna časová omezení vždy splněna [16].

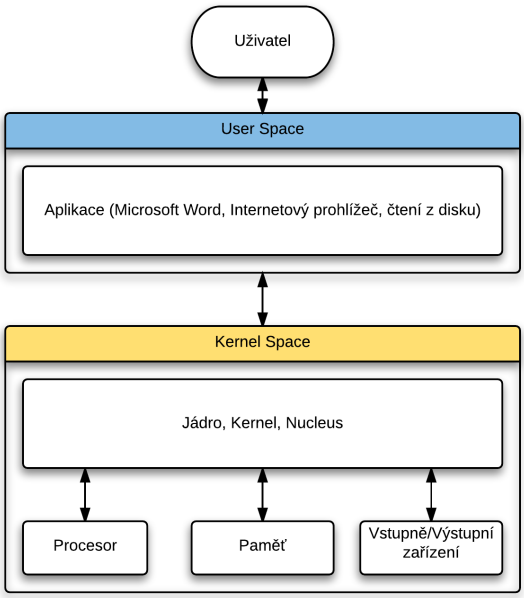
V obou systémech hraje důležitou roli plánovač. Plánovač musí zvládat plánování priorit a zpoždění spouštění úloh musí být malé. Čím nižší zpoždění plánovače, tím rychleji může být Real-Time úloha spuštěna [16].

Firm Real-Time operační systém

Firm Real-Time operační systém je kombinací hard a soft operačního systému. Úlohy mají kratší (soft) časové omezení a delší (hard) časové omezení [16].

1.2.4 Jádro operačního systému

Jádro, anglicky kernel nebo nucleus, je součástí operačního systému, které je zavedeno do chráněné oblasti paměti při startu počítače jako první. Tím je chráněno proti přepsání uživatelskými programy atd. Jádro je vrstva mezi hardwarem a softwarem počítače. Odstiňuje uživatele od nutnosti znát hardware počítače. Jádro vykonává svoje činnosti, jako například plánování, spouštění procesů a zpracovávání přerušení, v prostoru jádra (kernel space), zatímco uživatel vykonává svoje úlohy v uživatelském prostoru (user space). Uživatelská data a data jádra do sebe nezasahují. Oddělením těchto dvou prostorů je zajištěna větší stabilita systému a zvýšení rychlosti systému. Jádro běží po celou dobu běhu počítače, vyskytne-li se chyba v jádře a jádro přestane správně fungovat, počítač přestane fungovat. Jádro je klíčový prvek pro práci počítače. Vyskytne-li se chyba v user space aplikaci, je tato aplikace ukončena a počítač běží dále [26].



Obrázek 1.2: Vrstvy jádra [26].

Jádro zajištuje základní služby jako třeba správu paměti, správu procesů, správu souborů a správu vstupně-výstupních zařízení. Tyto služby jsou volány ostatními částmi operačního systému pomocí systémových volání. Systémové volání je poskytnuto rozhraním (API) jádra [26].

1.2.5 Přehled vícejádrových operačních systémů

Linux

Linux patří mezi Unix-Like operační systémy. Linux byl původně vyvinut Linusem Torvaldem v roce 1991 jako operační systém pro personální počítače firmy IBM založené na mikroprocesoru Intel 80386. Linus zůstává zapojen do vývoje Linuxu. Vývojáři během několika let zpřístupnili Linux na ostatní architektury jako například MIPS, PowerPC a SPARC. Zdrojový kód Linuxu je publikován pod licencí GNU GPL, a je tedy možné získat kompletní

plnohodnotný operační systém postavený nad Linuxem zdarma. Citováno z [40] původně převzato z [14].

Distribuce jsou sestavovány jednotlivci, týmy dobrovolníků, ale i komerčními firmami. Distribuce zahrnuje jádro, další systémový a aplikační software, grafické uživatelské rozhraní (X.org, KDE, GNOME) atd. Distribuce mají různá zaměření, například výběr obsažených programů, podpora určité počítačové architektury, použití ve vestavěných systémech atd. Mezi nejznámější distribuce patří Debian, Gentoo, Red Hat, Slackware a SUSE. Citováno z [40] původně převzato z [36].

Distribuce desktop Linuxu

Níže jsou uvedeny některé linuxové distribuce, které jsou vytvářeny pro stolní počítače.

- Antergos – Účelem této distribuce je nabídnout moderní, elegantní a výkonný operační systém založený na Arch Linuxu. Na rozdíl od Arch Linuxu není cílen pouze na zkušené uživatele nebo vývojáře, ale i na běžné uživatele. Instalace je grafická a uživatel si může vybrat jaké grafické prostředí (Gnome, Base, OpenBox, Cinnamon, XFCE, KDE) chce nainstalovat. Po instalaci již obsahuje všechny nutné aplikace pro běžnou práci [1].

- Mint – Je postavený na stabilní větvi Debianu a Ubuntu. Díky stabilní větvi Debianu je Mint stabilní systém určený spíše pro začínající uživatele. Stejně jako Antergos poskytuje po instalaci veškerý software potřebný pro běžnou činnost. Je zdarma a open source. Je k dispozici ve verzích s grafickým prostředím KDE nebo Cinnamon [3].

Embedded Linux

Vestavěný Linux obvykle označuje kompletní linuxovou distribuci, která je cílená na vestavěná zařízení. Neexistuje žádné jádro Linux, které je speciálně cíleno na vestavěná zařízení. Stejné jádro Linux je použito jak pro desktop, tak i pro vestavěná zařízení nebo Real-Time systémy. Citováno z [40] původně převzato z [42]. Existují i linuxová jádra, která jsou určena pro speciální typy vestavěných zařízení, například uCLinux (you-see-Linux). uCLinux dokáže na rozdíl od neupraveného Linuxového jádra běžet na procesorech bez MMU jednotky. Citováno z [40] původně převzato z [18].

Linuxové jádro může být zkompilováno pro velkou škálu zařízení, pro která umožňuje konfiguraci různých volitelných vlastností linuxového jádra. Vestavěné systémy s linuxovým jádrem se liší zejména použitou knihovnou jazyka C (musl, uClibc, glibc, dietlibc), což může mít značný vliv na velikost a výkon výsledného softwaru [40].

Distribuce Linuxu pro vestavěná zařízení

Níže jsou uvedeny známé linuxové distribuce, které jsou vytvořeny pro vestavěná zařízení.

- Embedded Debian je výrazně minimalizovaná verze hlavní distribuce Debian [5].

- Buildroot je nástroj, který zjednodušuje a automatizuje proces vytváření kompletního linuxového systému pro vestavěná zařízení pomocí cross kompilace. Buildroot je schopen vygenerovat toolchain na cross kompilaci, souborový systém root, obraz jádra Linux a bootloader na cílovou architekturu [15].

- Windriver je komerční Linux, který je optimalizovaný pro vestavěná zařízení. Tento Linux poskytuje out-of-the-box konfiguraci (po instalaci Linuxu není potřeba větších zásahů do konfigurace operačního systému), která usnadňuje programátorům tvořit a sestavovat jejich první projekty a aplikace. Citováno z [40] původně převzato z [6].
- Arch Linux Arm – Distribuce Arch, která poskytuje jádro a softwarovou podporu pro instrukční sady ARMv5te, hard-float ARMv6 a ARMv7, a ARMv8 AArch64. Arch je rolling-release distribuce, což znamená že systém je neustále aktualizován za běhu novými verzemi balíčků [7].

Android

Android je obsáhlá open source platforma vyvinuta speciálně pro mobilní zařízení prosazovaná společností Google a vlastněna Open Handset Aliancí, jejíž zakladatelem je společnost Google. Android poskytuje veškerý potřebný software pro mobilní zařízení. Je uživatelsky přívětivý a uživatel si jej může přizpůsobit dle svých představ [21].

Android je licencován pod Apache/MIT licencí, to znamená, že každému je umožněn přístup do zdrojových souborů celé platformy, které může svévolně rozšiřovat. Android je postaven na jádře Linux [21].

Windows

Windows je operační systém vytvořený společností Microsoft pro IBM kompatibilní počítače. Jádro tvoří operační systém MS-DOS. V pozdějších verzích bylo přidáno grafické uživatelské rozhraní. Vývoj tohoto operačního systému byl zastaven v roce 2001 kdy společnost Microsoft vyvinula nový systém se jménem Windows NT, který byl zpřístupněn veřejnosti již v roce 1993 (NT 3.0). Nyní je Windows NT nejpoužívanějším systémem na světě [22].

Windows si získal uživatele především díky jednoduchému ovládání a propracovanému grafickému rozhraní. Nyní má Microsoft Windows 88% podíl na trhu mezi desktopovými operačními systémy [2].

Mac OS X

Historie operačního systému Mac OS sahá do roku 1977, kdy firma Apple Computers vyprodukovala komerčně úspěšný počítač Apple II. Skutečný Mac OS 1.0 byl však uveden na trh až v lednu roku 1984 spolu s prvním počítačem Macintosh. Mac OS byl velmi pokrokovým operačním systémem, a kromě GUI obsahoval i další moderní prvky – ovládání myší, multitasking, multimédia, podporu práce v sítích atd. Mac OS byl do roku 2002 postupně uveden v 9 verzích. V roce 1994 Apple oznámil práce na zcela novém operačním systému s kódovým označením Copland. Na svou dobu měl Copland mnoho převratných designových prvků jako skutečné mikrojádro a hardwarovou abstrakci. Mac OS 8.6 představil multitasking na úrovni kernelu (jádra). V roce 1998 byl uveden Mac OS 9, který byl vyvíjen až do roku 2002, kdy byla jeho poslední verze 9.2 nahrazena zcela novým operačním systémem Mac OS X [4].

Mac OS X, který je na trhu od roku 2000, je moderní objektově orientovaný systém založený na kvalitním a stabilním základu BSD Unix, vybavený novým vektorovým grafickým rozhraním Aqua GUI. V pozadí nového uživatelského rozhraní stojí jádro OS Darwin, otevřená základna na bázi Unixu, postavená na takových technologiích jako Mach nebo

FreeBSD. Nad Darwinem/XNU stojí množina služeb a knihoven, které se starají o grafické rozhraní a uživatelské aplikace. Mac OS X nabízí kompletní implementaci systému X Window pro aplikace založené na X11. Z počítačů Apple Macintosh se tak postupně stává nejuniverzálnější platforma pro běžného i profesionálního uživatele [4].

1.2.6 Vícejádrová platforma

Dřívější vestavěné systémy se většinou skládaly z mikroprocesoru a několika příslušenství. Tyto systémy v průběhu své činnosti získaly malý objem dat, zpracovaly tyto data, vykonaly rozhodnutí na základě zpracovaných dat a na svůj výstup umístily informace na základě předchozích rozhodnutí. Moderní vestavěné systémy pracují s gigabity dat a provádějí analýzu celých datových sad. Často jsou na tyto systémy kladeny požadavky jako například podpora deterministických a operace s krátkou odezvou [41].

Vícejádrová platforma obsahuje procesorový systém, který implementuje více výpočetních elementů. Tyto výpočetní elementy se nazývají jádra procesorového systému. Každé z těchto jader může nezávisle vykonávat svoji posloupnost instrukcí a tím vytváří paralelní procesorový systém. Tento systém může pracovat s velkými shluky dat a provádět nad nimi různé typy analýz souběžně [41].

Odroid XU4

Odroid XU4 je vývojová vícejádrová platforma, která využívá ARM Big.LITTLE technologii a HMP řešení. Odroid-XU4 je nová generace počítačových systémů s velmi výkonným a úsporným hardwarem v malém provedení. Na platformě Odroid-XU4 lze spustit mnoho operačních systémů, například Ubuntu, Android či Xenomai RTOS [23].

Celkem platforma Odroid XU4 obsahuje 8 jader, která jsou obsažena v SoC Samsung Exynos 5422, který obsahuje 4 jádra typu Cortex A18 o frekvenci 2GHz a 4 jádra typu Cortex-A7. SoC sekunduje paměť o velikosti 4 GB typu LPDDR3 a grafický čip Mali-T628 MP6. Odroid obsahuje eMMC 5.0, USB 3.0 a gigabitové ethernetové rozhraní [23].

Jelikož Odroid XU obsahuje ve svém procesoru jádra různých archutektur, je na této platformě využíván princip heterogenního multiprocessingu, popsán v kapitole 1.4.3. Bližší informace k tomuto procesoru nejsou k dispozici. K získání technického manuálu je potřeba s vážným důvodem kontaktovat výrobce, tj. Samsung.

Raspberry Pi 3

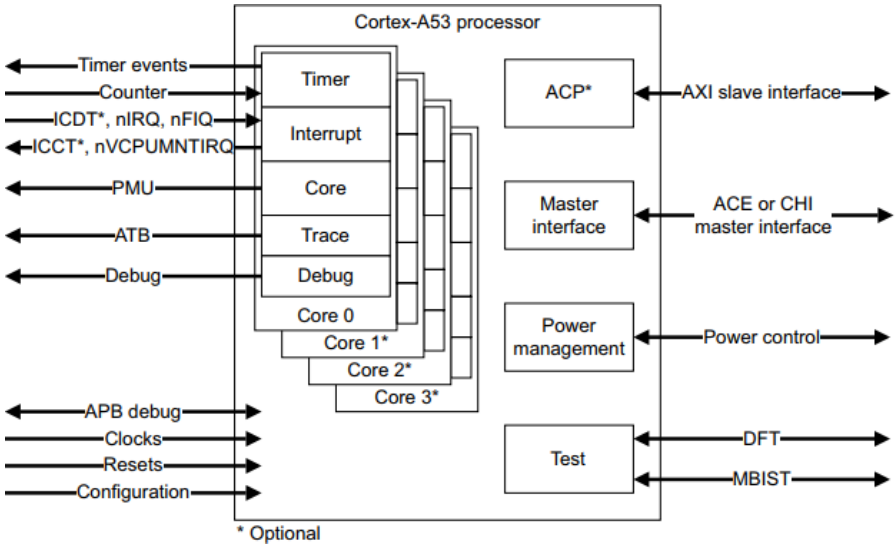
Raspberry Pi 3 je již třetí generace oblíbené platforma Raspberry. Obsahuje 64 bitový procesor typu ARM Cortex-A53 o frekvenci 1.2 GHz se čtyřmi jádry, kterému je dodávána paměť o velikosti 1 GB. O vykreslování se stará grafický čip VideoCore IV. Dále poskytuje rozhraní Bluetooth 4.1, Wi-Fi, HDMI, Ethernet, USB, rozhraní pro kamery a displeje [37].

Tato platforma je velmi oblíbená u začátečníků z důvodu její nízké pořizovací ceny, velmi kvalitní dokumentaci a obrovské komunity. Cena této platformy se pohybuje kolem 1000 korun [37].

ARM Cortex-A53

Tento procesor je založen na architektuře ARMv8-A. Obsahuje 4 jádra, přičemž každé jádro má svou vyrovnávací paměť úrovně L1 a svou jednotku pro správu paměti (MMU). Paměťový systém je založen na hardvardské architekruře. Procesor obsahuje jednu vyrovnávací

paměť úrovně L2 pro všechna jádra. Architektura ARMv8-A poskytuje podporu instrukční sady ARM, Thumb a A64. Jako rozšíření instrukčních sad lze využít například SIMD a Floating-Point rozšíření [8].



Obrázek 1.3: Blokový diagram procesoru ARM Cortex-A53 [8].

Dále tento procesor obsahuje generický přerušovací řadič verze 4 (GICv4) pro správu a podporu přerušení v systému. Poskytuje registry pro mapování zdrojů přerušení, chování přerušení a pro přerušení směrované více jádrům [8].

NXP Sabre

Vývojový kit NXP Sabre (Smart Application Blueprint for Rapid Engineering) je založen na multimediálně aplikačních procesorech rodiny i.MX. Platforma Sabre přináší podporu pokročilých technologií pro aplikace využívající tablety, elektronické čtečky nebo informační systémy. Sabre obsahuje spoustu hardwarových akcelerátorů a multimediálních kodeků spolu s širokou podporou nástrojů od firmy NXP [32].

Platforma Sabre pro chytrá zařízená založená na procesorech i.MX 6

Platforma je založená na dvou nebo čtyřjádrovém procesoru typu Cortex-A9 o frekvenci 1GHz a obsahuje paměť o velikosti 1 GB typu DDR3 SDRAM. Jako úložiště dat slouží 6 GB eMMC flash paměť. Na desce je umístěna SPI NOR flash paměť o velikosti 4 MB. Platforma obsahuje 10 palcový dotykový displej s rozlišením až 1024x768 a čtyři tlačítka. Pro obrazový výstup lze využít HDMI, LVDS, LCD a EPDC konektor. O správu napájení se stará čip od firmy NXP PMIC PF0100. Zvukový výstup je realizován skrze dva jednowatové reproduktory a nebo pomocí dvou 3.5mm konektorů. Na desce se nachází dva digitální mikrofonní vstupy. Dále podporuje ethernetové, SATA a USB rozhraní [33].

Vnitřní úložiště platformy je možné rozšířit připojením SD karty. Tato platforma je přímo určena pro chytrá zařízení, takže neschází ani podpora vložení baterky, GPS modul, dvě 5 megapixelové kamery, světelný senzor, akcelerometer atd. Platforma je cílena pro použití v oblasti internetu věcí, například jako senzor aktivity a wellness monitor nebo zdravotní systém [33].

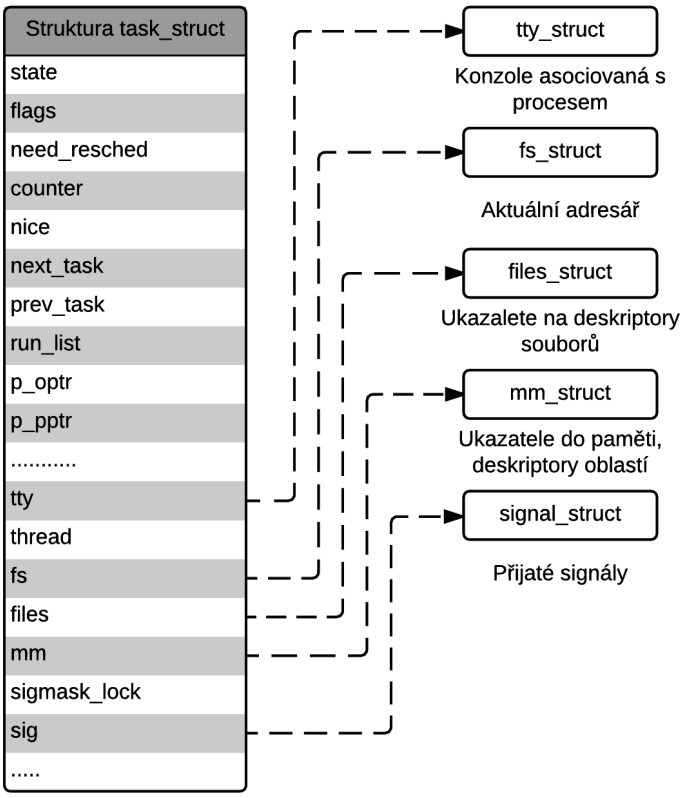
1.3 Základní funkce jádra a operačního systému

Mezi základní funkce operačního systému patří plánování a správa procesů, správa paměti a správa souborů.

1.3.1 Správa procesů

Proces je instancí vykonávaného programu. Vytvoří-li se nový proces, je téměř identický svému rodiči. Nový proces dostane logickou kopii adresového prostoru rodiče a začne vykonávat kód rodiče. Nový proces vykonává kód začínající na další instrukci za systémovým voláním (fork). Rodič a dítě (proces po forku) mohou sdílet stránky obsahující programový kód. Tyto adresové prostory se neprolínají, tudíž změny provedeny v synovském procesu se nepromítnou do rodičovského procesu a naopak [14].

Ke správě procesů jádro Linux využívá proces deskriptory. Deskriptor procesu je struktura typu task_struct obsahující všechny potřebné informace o jednom konkrétním procesu [14]. Mezi nejdůležitější informace o procesu patří stav procesu, priorita, přiřazenost úlohy jádru, informace o návratovém kódu, PID, ukazatel na rodiče, ukazatel na své synovské procesy, ukazatel na využívané procesy, Real-Time priorita, časovače, GID a signály spojené s procesem [43].

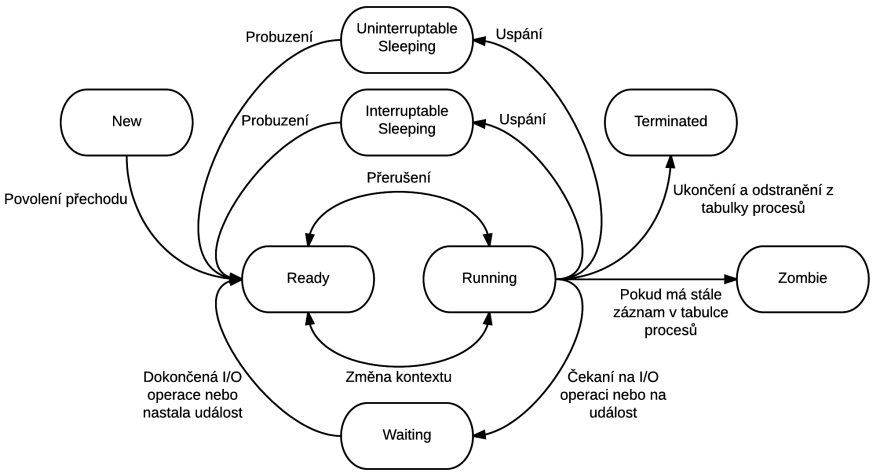


Obrázek 1.4: Popis struktury task_struct [14].

Stavy procesu

Každý proces v systému se musí nacházet v jednom z následujících stavů:

- New – Vytvořený proces, který čeká na povolení přechodu do stavu Running od plánovače úloh [14].
- Running – Proces, který má přidělen procesor [14].
- Waiting – Proces, který čeká na nějaké události systému, například dokončení vstupně-výstupní operace [14].
- Terminated – Proces, který dokončil vykonávání nebo byl ukončen. Citováno z [40] původně převzato z [29].
- Zombie – Proces, který dokončil vykonávání, ale má stále záznam v tabulce procesů. Citováno z [40] původně převzato z [19].
- Sleeping – Proces, který se vzdal procesorového času a změnil svůj stav na nespustitelný po celou dobu spánku. Jeho spánek lze vyrušit pomocí signálu SIGCONT. Citováno z [40] původně převzato z [34].
- Uninterruptible sleep – Proces, který se nachází v nepřerušitelném spánku. Citováno z [40] původně převzato z [31].



Obrázek 1.5: Diagram stavů procesů [40].

PID procesu

PID (Process ID) je unikátní identifikátor procesu v systému. Čísla procesů PID jsou číslována sekvenčně, tedy PID nově vytvořeného procesu je PID posledního vytvořeného procesu navýšen o jedničku. PID se nemůže zvyšovat do nekonečna, jádro Linux má horní limit (ve výchozím nastavení 32768) a po dosažení tohoto limitu jádro Linux začne recyklovat již použité PID pro nové procesy [14].

1.3.2 Plánování procesů

Plánování procesů v jádře Linux je založeno na sdílení procesorového času mezi více procesy. Každé jádro procesoru může vykonávat pouze jednu instrukci v daný čas. Pokud je proces vykonáván na procesoru a překročí své přidělené časové kvantum procesorového času, nastane odebrání procesoru tomuto procesu a procesor je přiřazen dalšímu procesu. Toto chování je obecné a liší se dle použitého plánovacího mechanismu. Tato přepnutí kontextu procesů využívají přerušení od časovače. Tímto je toto chování naprosto transparentní vůči procesům v systému, a tudíž není třeba nijak modifikovat kód spouštěných procesů. Plánovač také může využívat priorit procesů, které jsou v systému Linux dynamické [14].

Ve výchozím stavu je v jádře Linux využit preemptivní plánovač. To znamená, že je-li proces spuštěn a nachází se ve stavu Running, jádro Linux zkontroluje, zda je priorita nově spuštěného procesu vyšší než priorita již spuštěného procesu. Je-li priorita nově spuštěného procesů vyšší, jádro Linux přeruší vykonávání procesu a plánovač musí zvolit další proces ke spuštění [14].

Pokud jádro využívá nepreemptivní plánovač, musí plánovač počkat, dokud již běžícímu procesu nevyprší časové kvantum, anebo dokud se běžící proces procesoru nevzdá. Nepreemptivní plánovač nemůže přerušit běh běžící úlohy na procesoru [14].

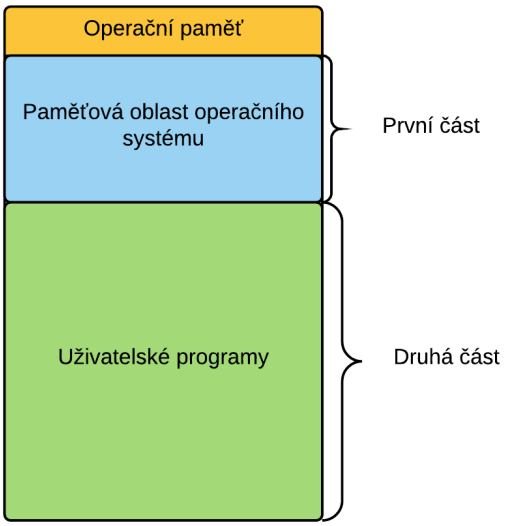
Plánovací mechanismy

Níže jsou popsány nejběžnější typy používaných plánovačů [38]:

- First-come First-served (FCFS) - Pokud je proces připravený k běhu, umístí se do fronty připravených úloh k běhu. Pokud již běžící proces ukončí vykonávání na procesoru, je z fronty připravených úloh vybrán proces, který přišel do fronty běžících úloh jako první.
- Shortest Job First scheduling (SJF) - Vybírá proces z fronty připravených úloh k běhu, který má nejmenší dobu následujícího procesorového cyklu. Má-li více procesů stejnou dobu následujícího procesorového cyklu, je mezi těmito procesy rozhodnuto pomocí algoritmu FCFS. Tento mechanismus je nepreemptivní.
- Shortest remaining time (SRT) - Preemptivní verze algoritmu SJF. Pokud je ve frontě připravených úloh proces s kratším následujícím procesorovým cyklem, než u aktuálně běžícího procesu, je běžící proces přerušen a umístěn zpět do fronty připravených úloh k běhu a začne se vykonávat proces s nejmenším následujícím procesorovým cyklem.
- Nepreemptivní plánování podle priorit – Každému procesu je přidělena priorita. Procesor je přidělen procesu s nejvyšší prioritou. Má-li více priorit stejnou prioritu, je vybrán proces podle mechanismu FCFS.
- Preemptivní plánování podle priorit – Preemptivní verze nepreemptivního plánování podle priorit.
- Round-Robin plánování – Round-Robin plánovač je založen na časovači. Přerušení od časovače je periodicky generováno. Pokud je přerušení vyvoláno, je aktuálně běžící proces uložen na konec fronty úloh připravených k běhu a z této fronty je plánovačem vybrán první proces, který bude vykonáván na procesoru (chování FCFS). Každému procesu je přiřazeno časové kvantum, po které běží na procesoru (perioda přerušení plánovače). Round-Robin plánovač je preemptivním mechanismem. Hlavním problémem je zvolení vhodného časového kvanta.

1.3.3 Správa paměti

Hlavní paměť je rozdělena na dvě části. První část obsahuje instrukce operačního systému a druhá část obsahuje úlohu, která je vykonávána uživatelem (jednoúlohový operační systém). Ve víceúlohovém operačním systému je tato druhá část dále rozdělována pro obsažení více úloh. Rozdělování paměti je realizováno dynamicky operačním systémem [38].



Obrázek 1.6: Rozdělení operační paměti [38].

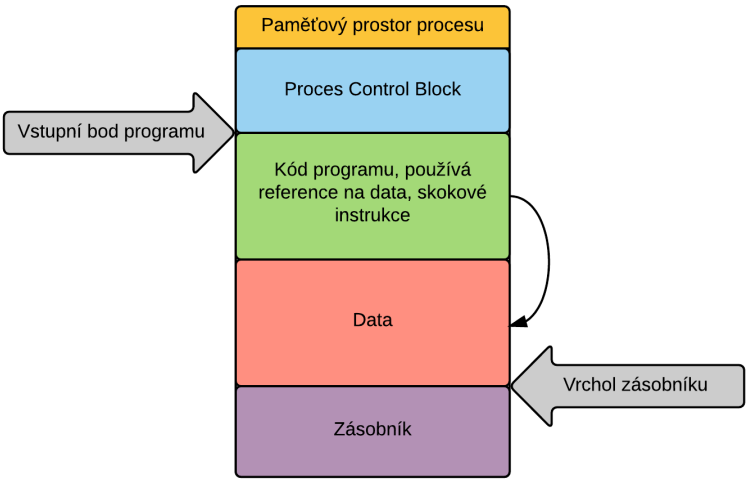
Paměť je logicky organizována jako jednodimenzionální nebo lineární adresový prostor, který obsahuje sekvenci bytů nebo slov. Sekundární paměť je organizována podobně. Většina programů je organizována jako moduly. Výhoda modulů spočívá v jejich možnosti být nezávisle zapsány, mohou být sdíleny mezi více procesy a ochrana paměti může být realizována na základě modulů [38].

Z hlediska fyzické organizace je paměť rozdělena na hlavní paměť, která je volatilní a poskytuje rychlý přístup při relativně vysoké ceně, a na nevolatilní sekundární paměť, která je levnější, ale pomalejší [38].

Relokace paměti

Relokace je základní služba správy paměti. Operační systém musí znát, kde se v paměti pro daný proces nachází [38]:

- Informace o procesu
- Zásobník procesu
- Vstupní body ke spuštění programu



Obrázek 1.7: Relokace paměti [38].

Sdílení paměti

Všechny mechanismy ochrany paměti musí povolit souběžný přístup několika procesům ke stejné oblasti v paměti. To znamená, že pokud více procesů vykonává stejný program, je výhodné, aby všechny procesy měly přístup do stejné oblasti paměti namísto toho, aby každý proces měl vlastní kopii této paměti [38].

Ochrana paměti

Každý adresový prostor procesu by měl být ochráněn a odstíněn od adresových prostorů jiných procesů [38].

1.3.4 Správa souborů

Soubor je kolekce informací definovaných uživatelem. Počítačový systém může ukládat soubory na úložiště (pevný disk, optický disk, přenositelné uložiště). Soubory jsou většinou uloženy v adresářích [39].

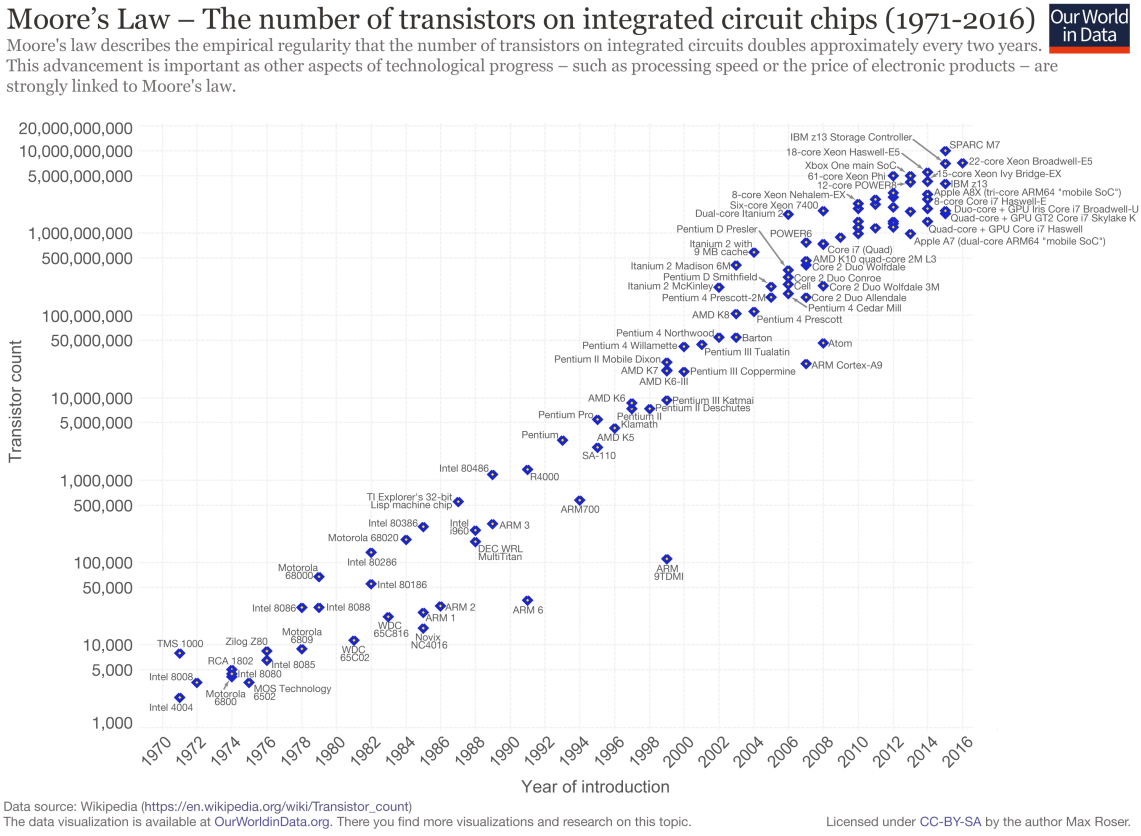
Základní služby správy souborů jsou [39]:

- Vytváření a mazání souborů a adresářů.
- Podpora pro manipulaci se soubory a adresáři (základní operace: přejmenování, přístupová práva atd.).
- Mapování souborů na sekundární úložiště.

1.4 Řešená problematika

Jak technologický pokrok poskytnul prostředí pro rozšiřující se Real-Time vestavěné systémy, nároky na stále pokročilejší a sofistikovanější funkce se každý rok zvyšovaly. Až do

nedávna byl stabilní pokrok ve výrobě polovodičů schopen držet krok s neustále se zvyšujícím počtem tranzistorů na čipu, což vedlo k obrovskému nárustu těchto polovodičů na čipu. Nicméně miniaturizace již dosáhla svého limitu. Vzhledem k neschopnosti regulace energetických a teplotních ztrát při vysokých frekvencích vývoj jednojádrových procesorů stagnuje [24].



Obrázek 1.8: Moorův zákon - Počet transistorů se s každým rokem zvyšuje při zachování ceny [35].

Po zastavení vývoje jednojádrových procesorů museli výrobci přijít s novou strategií. Místo zvyšování výkonu jednojádrového procesoru vznikl nápad přidat další jádra na procesor. Od této doby začal počet jader na procesor růst a tento trend platí i dnes. První vývojová platforma, která se objevila, obsahovala pouze málo jader a byla pojmenována jako vícejádrová platforma (multi-core platform). Platforma s více než deseti jádry byla pojmenována mnohojádrová platforma (many-to-core platform) [24].

1.4.1 Paralelismus na úrovni instrukcí

MIMD - Multiple Instruction Multiple Data

V MIMD architektuře pracuje několik procesorů paralelně a asynchronně. Většinou tyto procesory sdílí přístup do běžné paměti [24].

Každý procesor vykonává svou vlastní sekvenci instrukcí, pracuje na jiné části problému a předává data svým sousedům. Procesory můžou čekat na data od svých sousedů nebo čekat na přístup do paměti [24].

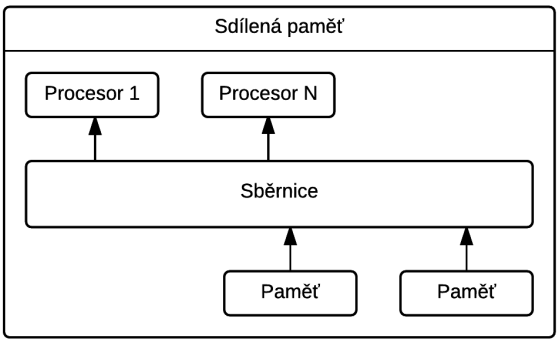
SIMD - Single Instruction Multiple Data

U paralelismu typu SIMD ovládací jednotka (CU) načítá a dekóduje instrukce. Tyto instrukce jsou poté buď přímo zpracovány v CU samotné (skoky atd.), nebo jsou rozeslány kolekci výpočetních jednotek (Processing elements). Tyto procesní jednotky vykonávají svou práci synchronně, ale jejich lokální paměti se od sebe liší. Vzhledem ke složitosti CU jednotek, výpočetního výkonu, adresovacích metod výpočetních jednotek a propojením mezi výpočetními jednotkami můžeme rozlišovat několik různých podtříd zařízení SIMD [24].

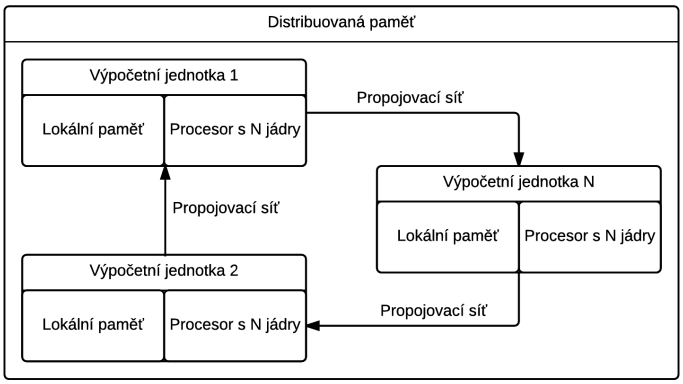
SIMD pracuje s velkým počtem jednoduchých procesorů, kde je využit datový paralelismus, který přidělí procesor jednotlivým úsekům dat. Typicky jsou využity distribuované paměti z čehož plyne, že se v SIMD zařízeních mohou vyskytovat komunikační problémy [24].

1.4.2 Sdílená a distribuovaná paměť

Při použití sdílené paměti mají všechny procesory přímý přístup do všech pamětí. Při použití distribuované paměti má každý procesor přístup pouze do své lokální paměti. SIMD využívá distribuované paměti a MIMD sdílené paměti [24].



Obrázek 1.9: Sdílená paměť [35].



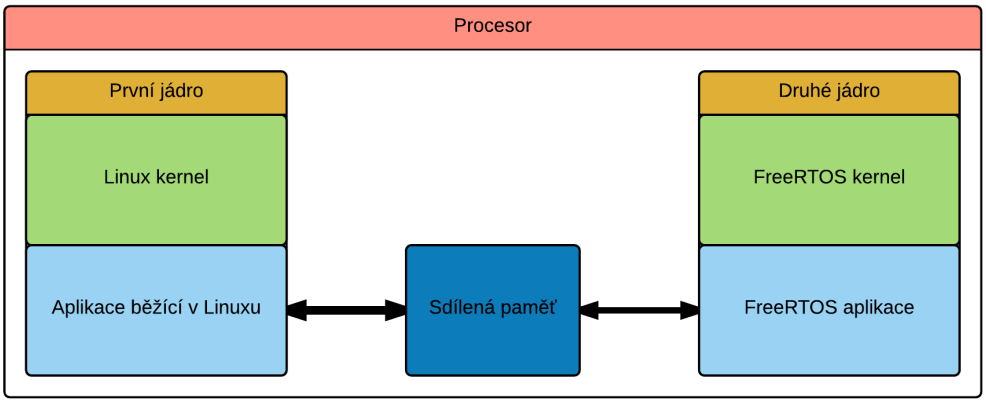
Obrázek 1.10: Distribuovaná paměť [35].

1.4.3 Vícejádrová architektura

Ve vícejádrových procesorech se využívá několika přístupů multiprocessingu AMP, HMP a SMP. AMP je Asymetrický Multi-Processing, SMP je Symetrický Multi-Processing a HMP je Heterogenní Multi-Processing. Všechny tyto přístupy jsou popsány níže.

Asymetrický Multi-Processing

Asymetrický Multi-Processing má několik procesorů, které nemusí mít stejnou architekturu. Každý z těchto procesorů má svůj vlastní adresový prostor a každý z těchto procesorů může nebo nemusí být aktivní. AMP se používá při úlohách, které jsou optimální pro různé procesorové architektury, jako například DSP nebo MCU. Při používání AMP systému je možné spustit různé operační systémy na různých jádrech (například FreeRTOS a Linux) [45].



Obrázek 1.11: Příklad AMP systému [45].

Heterogenní Multi-Processing

Heterogenní Multi-Processing systém se skládá z několika jednojádrových nebo vícejádrových procesorů různých typů. Nejjednodušší formou heterogenního systému je kombinace vícejádrového procesoru s grafickým čipem [41].

Nicméně dnešní technologie dovolují heterogenním vícejádrovým systémům na čipu obsahovat například [41]:

- Vícejádrové aplikační procesory.
- Vícejádrové grafické procesory.
- Vícejádrové real-time procesory.
- Jednotku pro správu platformy.
- Jednotky pro konfiguraci a zabezpečení platformy.
- Několik vícejádrových procesorů implementovaných v programovací logice FPGA čipu.

Evoluce Heterogenních výpočetních systémů s FPGA

V roce 2002 společnost Xilinx představila FPGA obsahující aplikační procesor PowerPC 405. Poté společnost Xilinx vyvinula další generace tohoto FPGA s výkonnějšími PowerPC procesory a dvěma PowerPC procesory na čipu. Namísto dnešních generací zařízení je výpočetní systém integrovaný ASSP, tj. výpočetní systém obsahuje procesor, interconnect, řadič paměti a periferie. Tyto první generace potřebovaly významný počet FPGA zdrojů, aby mohly být považovány jako ASSP výpočetní systém [41].

V roce 2011 Xilinx vyvinul rodinu procesorů typu Zynq-7000, které obsahují plně integrované výpočetní zařízení s dvoujádrovým procesorem typu Cortex-A9 společnosti ARM, interconnectem, řadičem paměti a periferiemi zkombinovaný s programovatelnou logikou založenou na Xilinx FPGA řady 7 [41].

Jedna z výhod použití programovací logiky FPGA k implementaci vícejádrových procesorů je možnost paralelizace zpracovávání dat ve dvou dimenzích pomocí paralelních rour a několikati fází zřetězení. Tím lze docílit vysokého počtu zpracovaných instrukcí za takt [41].

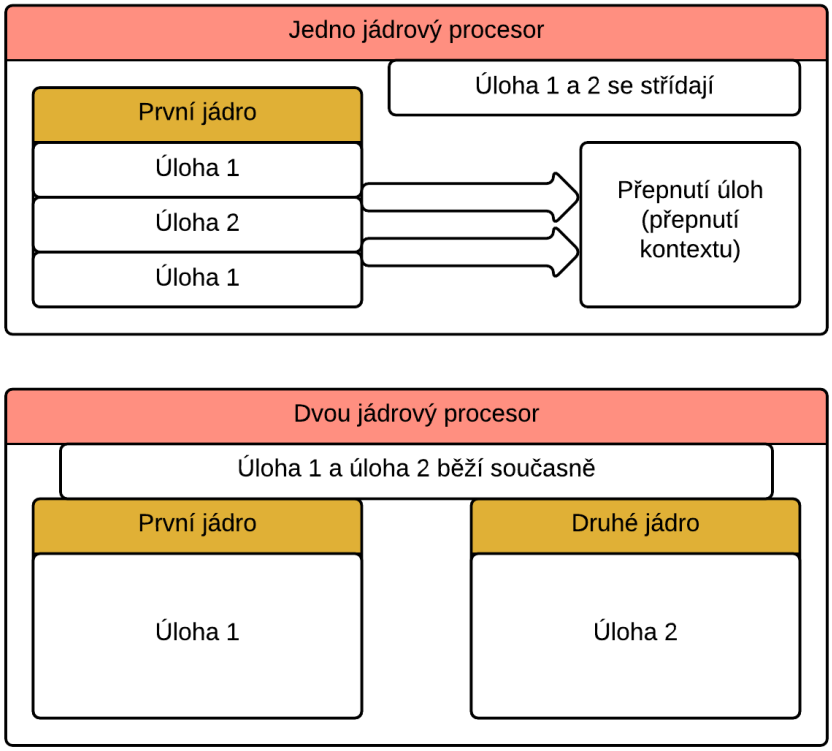
Symetrický Multi-Processing

Symetrický Multi-Processing má také několik procesorů, od AMP se ale liší v tom, že tyto procesory musí mít stejnou architekturu. Musí to být homogenní vícejádrový design. Procesory sdílí adresový prostor nebo nějakou jeho část. Většinou je operační systém spuštěn na všech procesorech a rozděluje práci mezi jádra. Komunikace mezi jádry je zařízena pomocí sdílené paměti, která je přístupná díky rozhraní operačního systému [45].

Povinností operačního systému je plánovat procesy, které mají běžet má každém procesoru v systému a poskytovat komunikační a synchronizační mechanismy mezi těmito procesy. Plánování procesů na více než jednom procesoru zvyšuje komplexnost plánované úlohy. Místo toho, aby plánovač rozhodnul, co plánovat, musí také rozhodnout na jaký procesor danou úlohu plánovat [12].

1.4.4 Plánování ve vícejádrovém systému

Na rozdíl od jednojádrového systému je ve vícejádrovém systému spuštěno souběžně tolik úloh, kolik jader obsahuje procesor. Tato jádra pracují nezávisle na sobě a každé jádro zpracovává svoji instrukci. V jednojaderném systému je toto chování pouze emulováno, ve skutečnosti jsou úlohy mezi sebou rychle přepínány tak, aby utvořily dojem souběžného běhu. Při programování aplikací pro vícejádrové systémy je nutné dbát na práci se sdílenými prostředky, v jednojádrovém systému stačí při práci se sdílenou proměnnou vytvořit kritickou sekci, která zakáže přerušení a tím je zaručen výlučný přístup ke sdílené proměnné. U vícejádrových systémů je však tento problém složitější z důvodu více jader. Je nutné zachovat výlučný přístup ke sdílenému zdroji napříč všemi jádry.



Obrázek 1.12: Porovnání plánovaní v jednojádrovém a vícejádrovém operačním systému

Přepínání kontextu

Přepínání kontextu úlohy se skládá z pěti základních kroků [27]:

- Vyvolání přepnutí kontextu pomocí například přerušení (preemptivní plánovač).
- Uložení kontextu běžící úlohy. V tomto kroku je uložen obsah registrů procesoru a obsah registru programového čítače. Tím je uložena pozice instrukce, kterou procesor prováděl před přepnutím kontextu nebo již následující instrukce.
- Plánovač vybere novou úlohu k běhu.
- Obnova kontextu nové úlohy. Nahraje uložený kontext následující úlohy do procesoru a tím je zaručen opětovný start programu od naposledy provedené instrukce (obsah registru programového čítače).
- Pokračování ve vykonávání nové úlohy.

Plánovací principy ve vícejádrovém systému

Plánování ve vícejádrovém prostředí se dělí na tři základní typy plánování [17]:

- Rozdělovací plánování
- Globální plánování
- Hybridní plánování

Rozdělovací plánování

N úloh je rozděleno mezi m procesorů. Na každém procesoru je prováděno plánování pomocí zvoleného plánovacího mechanismu. Algoritmické rozdělení úloh mezi m procesorů je NP těžký problém. Je-li počet procesorů v systému známý, nastává vyhledávácí problém (knapsack problem), v opačném případě optimalizační problém (bin-packaging problem) [17].

Mezi optimální plánovací mechanismy pro jednotlivé procesory patří RM, DM, EDF a LLF. Plánovací mechanismy pro bin-packaging heuristiku jsou například best fit (BF), first fit (FF), worst fit (WF), next fit (NF) atd. Rozdělovací plánovací mechanismus se skládá z plánovacího mechanismu určeného pro procesor a plánovacího mechanismu určeného pro bin-packaging heuristiku [17].

Mezi výhody tohoto plánování patří nezávislost jednotlivých plánovacích algoritmů, žádná migrační cena, na jednotlivé procesory lze přímo používat testy plánovatelnosti a v případě přetížení systému jsou jednotlivé procesory od sebe izolovány [17].

Globální plánování

Využívá globálního plánovacího mechanismu a jedné fronty úloh připravených k běhu. Při každém rozhodnutí plánovač rozhoduje, kam a kde bude plánovat m úloh. Migrace úloh je povolena [17].

Mezi výhody patří přizpůsobitelnost pro dynamické konfigurace vícejádrových systémů, existují optimální plánovače a přetížení/nevytížení procesorů je rozšířeno rovnoměrně mezi všechna jádra. Mezi nevýhody patří migrace úloh a nutnost použití synchronizačních prostředků pro zajištění výlučného přístupu ke frontě úloh připravených k běhu [17].

1.4.5 Správa paměti ve vícejádrovém systému

Kolize v přístupu do paměti (Memory Contention)

Kolize v přístupu do paměti nastane tehdy, pokud se pokusí dva různé programy použít stejný paměťový zdroj, jako disk, RAM, cache ve stejný čas. Tento jev může vyústit v deadlock nebo v neustálé stránkování paměti (memory thrashing). Přístup ke sdílené paměti by tedy měl být regulován čekáním nebo použitím správného plánovacího mechanismu [34].

Kolize v přístupu na hromadu (Heap Contention)

Hromada je reservovaná oblast v paměti, kterou aplikace mohou použít k uložení dočasných dat a je sdílena mezi procesorovými jádry během vykonávání programu. Heap Contention je jedna z nepříjemností při vykonávání vícejádrových aplikací, které vyžadují intensivní alokaci paměti. Tento problém může být vyřešen vytvořením privátní hromady. Využití privátní hromady také zvýší výkon vícejádrového systému [34].

Falešné sdílení

Falešné sdílení nastane, pokud dva nebo více procesorů ve vícejádrovém sytému využívají stejný řádek ve vyrovnávací paměti, který není souběžně spojen/nesouvisí s prováděnými operacemi. Systém vyrovnávací paměti začne invalidovat a přepisovat načtené kopie záznamů vyrovnávací paměti ostatním jadrům [34].

Každý procesor může mít jiné mechanismy, jak předcházet falešnému sdílení. Falešné sdílení může být předcházeno opatrným zarovnáním datových struktur tak, aby datové struktury splňovaly hranice řádku ve vyrovnávací paměti použitím kompilátorového zarovnávacího programu datových struktur pro každý procesor. Jiná cesta, jak předcházet falešnému sdílení, je seskupování často používaných polí dat, aby se zajistilo, že budou uloženy ve stejné vyrovnávací paměti a lehce přístupné [34].

Kolize při zamykání (Lock Contention)

Kolize při zamykání vzniká, pokud se vlákno pokusí zamknout program, který je již získán jiným vláknem. Tento spor může být vyřešen adopcí lock-free algoritmů a konkurentních datových struktur, které eliminují zámky a synchronizační nástroje jako například mutex. Konkurentní datové struktury nevyžadují synchronizační mechanismy [34].

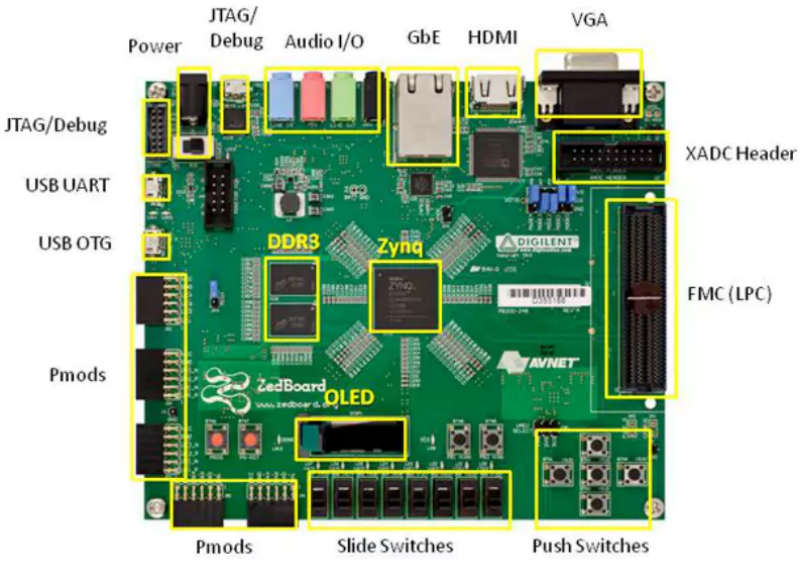
Kapitola 2

Zvolené realizační prostředky

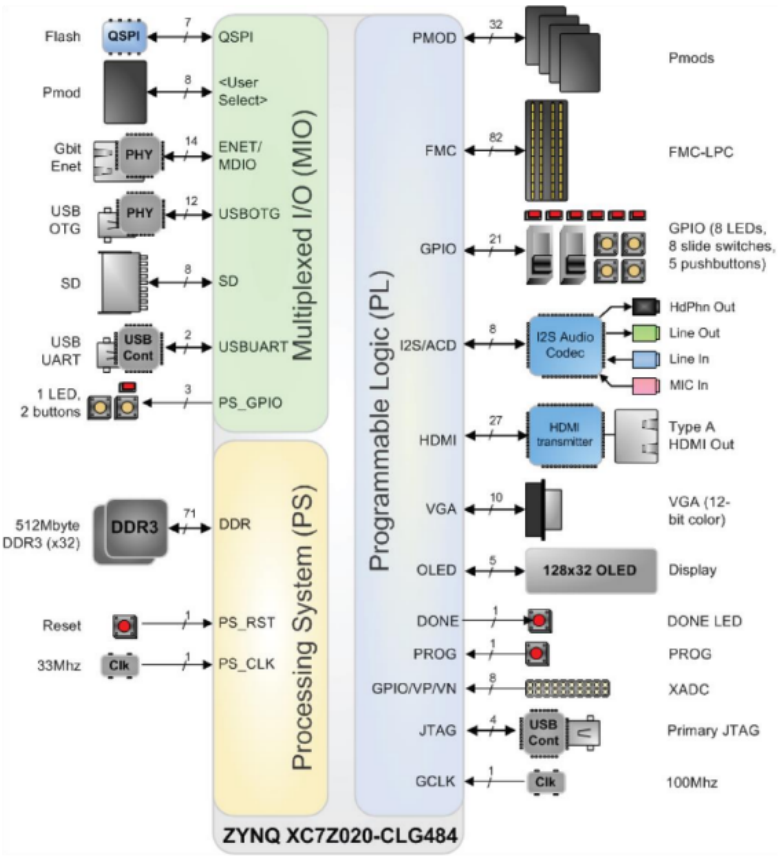
2.1 ZedBoard

ZedBoard je cenově dostupný vývojový kit s procesorem Xilinx Zynq-7000 All Programmable SoC. Platforma obsahuje vše potřebné pro vytvoření designů založených na Linuxu, Androidu, Windows nebo ostatních operačních a Real-Time operačních systémech. Několik rozšiřovacích konektorů odhaluje vstupně-výstupní porty procesorového systému a programovací logiky SoC Zynq-7000 pro snadné použití. ZedBoard je podporován zedboard.org komunitou [11].

ZedBoard obsahuje SoC Zynq-7000, paměť DDR3 o velikosti 512 MB a Quad SPI Flash paměť o velikosti 256Mb. Pro komunikaci s vývojovým kitem lze použít dostupné USB-JTAG rozhraní, ethernet rozhraní o rychlosti 1Gbps a USB-UART nebo USB-OTG rozhraní. Dále obsahuje rozšiřující konektor FPM-LPC, headery 5Pmod a Agile Mixed Signaling header. O časování se stará 100MHz oscilátor pro programovací logiku a hodiny o frekvenci 33,33333 MHz pro procesorový systém. Pro grafický výstup lze využít na desce integrovaný OLED displej nebo rozhraní VGA a HDMI. ZedBoard také obsahuje GPIO sestávající s osmi LED diod, 7 spínačů a 8 DIP přepínačů [11].



Obrázek 2.1: Periferie platformy Zynq-7000 [30].



Obrázek 2.2: Blokový diagram hardwaru kitu ZedBoard [11].

Konfigurace ZedBoard pro start systému za pomocí SD karty

Pro nastartování systému pomocí SD Karty musí být piny JP9 a JP10 připojeny na napětí 3,3 voltů. Dále musí být zajištěno propojení pinů JP6. Pro komunikaci s vývojovým kitem lze využít například rozhraní USB-UART. Při použití hostovacího systému Linux se ZedBoard hlásí jako zařízení /dev/ttyACM0. Pro komunikaci není třeba použít speciální kabel který konvertuje sériové rozhraní UART na USB-UART, ZedBoard má na desce konvertor FS232R, který tuto konverzi provádí [10].

Standardní nastavení sériové komunikace je [10]:

- Rychlost přenosu: 115200 bitů za sekundu
- Počet datových bitů: 8
- Parita: žádná
- Počet stop bitů: 1
- Řízení toku: žádné

Konfigurace ZedBoard pro start systému za pomocí JTAG rozhraní

Všechny piny JP7-JP11 musí být uzemněny (přivedeny na GND pomocí propojky). Pro naprogramování kitu je nutné použít USB-JTAG rozhraní. Pro komunikaci lze opět využít USB-UART rozhraní [10].

2.1.1 Procesor ARM

Procesor ARM (Acorn Risc Machine) je procesor s redukovanou instrukční sadou (RISC procesor). Koncept RISC byl vytvořen v laboratořích na Standfordu a Barkeley okolo roku 1980. ARM byl poprvé vyvinut v Acorn Computers Limited na Cambridge mezi roky 1983 a 1985. Byl určen pro komerční účely a lišil se v některých bodech od typické RISC architektury. V roce 1990 byla založena firma ARM Limited, která byla určena speciálně pro rozšíření technologie známé jako ARM. Brzy ARM ovládnul světové trhy s nízko-příkonovými a cenově dostupnými vestavěnými zařízeními [20].

Jedinými příklady RISC architektury byly v té době Berkeley RISC I a II a Standford MIPS. V architektuře ARM je zahrnuto několik vlastností Berkeley RISC designu.

Byly použity následující vlastnosti designu Berkeley RISC [20]:

- Load-Store Architektura.
- Fixní 32-bitové instrukce.
- Tříadresní formát instrukcí.

2.1.2 Xilinx Zynq-7000 All Programmable SoC

Rodina Zynq-7000 je založena na Xilinx All Programmable SoC architektuře. Tyto produkty kombinují jednojádrový nebo vícejádrový ARM Cortex A9 MPCore procesorový systém s programovatelnou logikou firmy Xilinx v jediném zařízení vytvořeném pomocí vysoce výkonové, nízkopříkonové, 28 nanometrové technologie. ARM Cortex A9 MPcore procesory jsou srdcem procesorového systému, který obsahuje paměť na čipu, externí paměťové rozhraní a bohatou množinu vstupně-výstupních periferií. Zynq-7000 nabízí flexibilitu a škálovatelnost FPGA pole a zároveň poskytuje výkon a snadné použití, které je typicky spojováno s ASSP a ASIC obvody. Na platformě Zynq-7000 lze vytvářet řešení s důrazem na vysoký výkon, ale i na malou cenu. Zynq-7000 SoC jsou používány v mnoha zařízeních, například v kamerách, při zpracovávání videa, v LTE sítích atd. [10].

Při inicializaci SoC Zynq-7000 jsou vždy procesory v procesorovém systému nastartovány jako první a tím dovolují start a konfiguraci programovací logiky. Konfigurace programovací logiky může být součástí startovacího procesu nebo může být nakonfigurována později. Programovací logika může být kompletně rekonfigurována nebo použita s částečnou nebo dynamickou rekonfigurací. Tyto rekonfigurační přístupy umožňují přizpůsobování programovací logiky na základě prováděných akcí. Konfigurace programovací logiky je označována jako bitstream. Procesorový systém a programovací logika jsou připojeny na oddělené napájecí větve a tím umožňují uživateli vypnout celou programovací logiku pro úsporu energie [10].

Zynq-7000 AP SoC je složen z následujících komponent [10]:

- Procesorový systém
  - APU – Aplikační procesorová jednotka
  - Rozhraní paměti
  - Vstupně-výstupní periferie
  - Interconnect
- Programovací logika

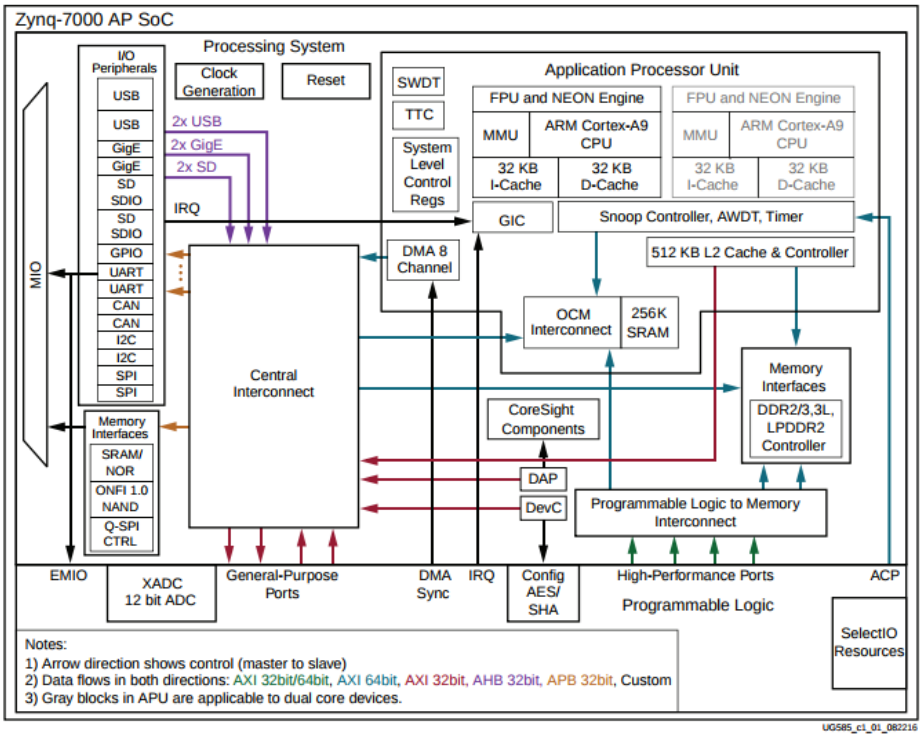


Figure 1-1: **Zynq-7000 AP SoC Overview**

Obrázek 2.3: Diagram platformy Zynq-7000 [49].

Programovací logika založená na FPGA řady Xilinx 7 (Artix a Kintex) je použita pro rozšíření funkcionality pro dosažení specifických aplikačních požadavků.

Programovací logika Zynq-7000 obsahuje několik typů FPGA zdrojů, například [10]:

- Konfigurovatelné logické bloky (CLBs)
- Konfigurovatelná bloková paměť (BRAM)
- Uživatelsky konfigurovatelný analogově digitální převodník (XADC)
- PCIe blok
- Konfigurovatelný blok s 256 bitovým AES pro dešifrování a SHA pro autentizaci

Vstupně-výstupní periferie včetně rozhraní statické/flash paměti sdílí multiplexované výstupně výstupní piny (MIO). Počet těchto MIO pinů je maximálně 54. Zynq-7000 také podporuje možnost použít vstupně-výstupní porty, které jsou součástí programovací logiky pro mnoho vstupně-výstupních periferií procesorového systému. Tato vlastnost je dosažena pomocí rozšířeného multiplexovaného vstupně-výstupního rozhraní (EMIO) [10].

Start SoC Zynq-7000 desky ZedBoard

SoC může být nastartován pomocí zabezpečeného i nezabezpečeného startu, v obou případech lze konfigurovat programovací logiku. Oba tyto startovací přístupy využívají bloky pro dešifrování AES a autentizaci pomocí SHA, které jsou součástí programovací logiky. Pro využití těchto přístupů musí být programovací logika napájena [10].

Startovací proces je rozdělen do více kroků a vždy zahrnuje startovací ROM paměť a FSBL (First Stage Boot Loader) zavaděč. Zynq-7000 obsahuje ROM paměť se základním zavaděčem. Tato paměť není přístupná uživateli. Zavaděč v bootROM detekuje, zda bude prováděn zabezpečený nebo nezabezpečený start systému, vykoná počáteční inicializaci sytému a přečte nastavení pinů na desce ZedBoard tak, aby rozhodnul, ze kterého zařízení bude ZedBoard startovat (SD karta, Jtag, QuadSPI). Po dokončení inicializace je spuštěn FSBL zavaděč z detekovaného startovacího média [10].

2.1.3 Procesorový systém Zynq-7000

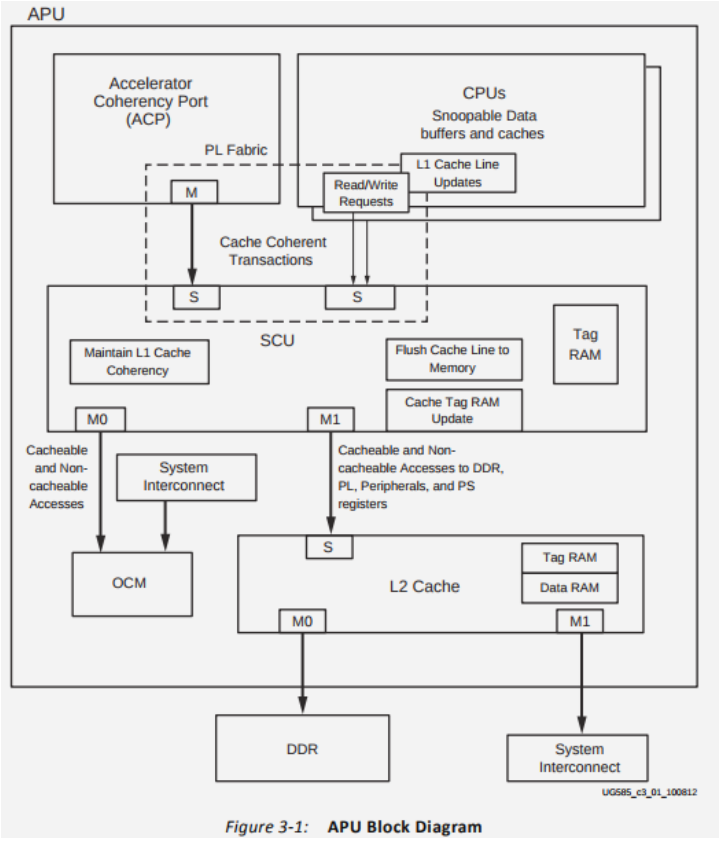
Aplikační procesorová jednotka – APU

APU implementuje dvoujádrovou Cortex-A9 MP konfiguraci architektury armv7h, která umožňuje asymetrické a symetrické multiprocesorové konfigurace. Každé jádro obsahuje 128 bitový SIMD koprocesor, VFPv3 (akcelerace vektorových operací), jednotku pro správu paměti (MMU) a oddělené instrukční a datové vyrovnávací paměti na úrovni L1 o velikosti 32Kb. Obě jádra poskytují dvě 64bitové AXI Master rozhraní pro nezávislé instrukční a datové transakce do SCU. V závislosti na adrese a atributech jsou tyto transakce směrovány do paměti na čipu (OCM), vyrovnávací paměti úrovně L2, DDR paměti nebo pomocí interconnectu v procesorovém systému do ostatních zařízení v procesorovém systému nebo programovací logice. Rozhraní každého jádra s SCU obsahuje nutné snoop signály k udržení koherence mezi vyrovnávacími pamětmi na úrovni L1 ve všech jádrech a sdílenou vyrovnávací pamětí o velikosti 512Kb na úrovni L2. Každé jádro má privátní časovače a watchdog [10].

Vlastnosti procesoru ARM Cortex A9 [10]:

- ACP (Accelerator Coherency Port) - Zprostředkovává komunikaci mezi programovací logikou a aplikační procesorovou jednotkou.
- SCU (Snoop Control Unit) - Jednotka pro udržení konzistence L1 a L2 cache pamětí, 2x AXI Master rozhraní
- Paměť SRAM s paritou o velikosti 256 Kb na úrovni L2 (přístupná procesoru, programovací logice a interconnectu).
- Řadiče DMA – Čtyři pro procesorový systém (kopírování paměti v systému) a čtyři pro programovací logiku (kopírování z programovací logiky do paměti a zpět).

- GIC (General Interrupt Controller) - Individuální přerušovací masky a prioritizace přerušení.
- Časovač watchdog, tři čítače/časovače.
- Řadič DDR – Podpora DDR3, DDR3L, DDR2, DDR2L.
- Řadič Quad SPI – Možnost startování pomocí tohoto rozhraní.
- Řadič statické paměti – Možnost startování pomocí tohoto rozhraní, řadiče pro NAND a paralelní SRAM/NOR řadič.
- GPIO - 54 GPIO signálů směrovaných pomocí MIO; 192 GPIO signálů mezi procesorovým systémem a programovatelnou logikou pomocí EMIO; funkce každého GPIO pinu lze nakonfigurovat dynamicky
- Dva řadiče ethernetu o rychlosti 1Gbps.
- Dva řadiče USB (USB 2.0).
- Dva řadiče SD/SDIO – Možnost startování pomocí SD karty.
- Řadiče sběrnic 2x SPI, 2x CAN, 2x UART a 2x I2C.



Obrázek 2.4: Blokový diagram APU [49].

Centrální výpočetní jednotka - CPU

Každé jádro dokáže zpracovávat dvě instrukce v jednom cyklu a provádět je mimo pořadí (out of order). CPU implementuje dynamickou predikci skoků (GHB, BTAC). Cortex-A9 je založen na armv7h architektuře s plnou podporou virtuální paměti a dokáže provádět 32 bitové ARM instrukce, 16 a 32 bitové Thumb instrukce a 8 bitový Java byte kód v Jazelle hardwarově akcelerovaném stavu [10].

Cache na úrovni L1

Vyrovnávací paměti na úrovni L1 mohou být nezávisle zakázány pomocí koprocesoru ovládajícího systém (system control coprocessor/system control register), jsou čtyř cestně asociativní, což je hybridní model mezi plně asociativní cache a přímo mapovanou cache. Tyto vyrovnávací paměti podporují virtuální stránky o velikosti 4KB, 64KB, 1 MB a 16 MB. Jako algoritmus pro nahrazování stránek je zvolen round-robin, obě L1 cache paměti podporují paritu. Po restartu CPU je obsah obou L1 cache pamětí vyčištěn [10].

Inicializace L1 cache paměti

Před použitím L1 pamětí je nutné invalidovat instrukční vyrovnávací paměť, datovou vyrovnávací paměť a BTAC (vyrovnávací paměť adres cílů skoků). Invalidovat hlavní TLB (tabulka stránek) tabulku není třeba [10].

Kroky nutné k inicializaci L1 cache [10]:

```
; Instrukce msr provede presun ARM registru do koprocesoru
; mcr op1{cond} coproc, #opcode1, Rt, CRn, CRm{, #opcode2}
; r0 = 0
; Invalidace TLB tabulek
mcr p15, 0, r0, c8, c7, 0
; Invalidace insktrukcni cache
mcr p15, 0, r0, c7, c5, 0
; Invalidace pole prediktoru skoku
mcr p15, 0, r0, c7, c5, 6
; Invalidace datove cache
mcr p15, 0, r11, c7, c14, 2
; Inicializace MMU
; Povoleni datove a instrukcni cache, r0 = 0x1004
mcr p15, 0, r0, c1, c0, 0
; Datova synchronizace barier, DSB zpusobi, ze zadna instrukce
; pred touto instrukci nebude provedena drive
; nez nasledujici instrukce
dsb
; Instrukcni synchronizace barier, ISB zpusobi, ze zadna instrukce
; pred touto instrukci nebude provedena drive
; nez nasledujici instrukce
isb
```

Inicializace CPU

Pro inicializaci jádra CPU je nutné provést následující kroky [10]:

- Nastavení bázové adresy vektorové tabulky (VBAR – Vector Base Address Register)
- Inicializace L1 cache pamětí viz. 2.1.3
- Invalidace L2 cache paměti
- Příprava tabulky stránek a její nahrání do paměti
- Inicializace zásobníku
- Nahrání bázové adresy tabulky stránek do TTBR (Translation Table Base Register)
- Nastavení povolovacího bitu jednotky MMU v system control registru
- Inicializace a povolení L2 cache pamětí
- Povolení L1 cache pamětí zapsáním do system control registru
- Skok na adresu začátku aplikace

Programovací logika – FPGA

Programovací logika poskytuje [10]:

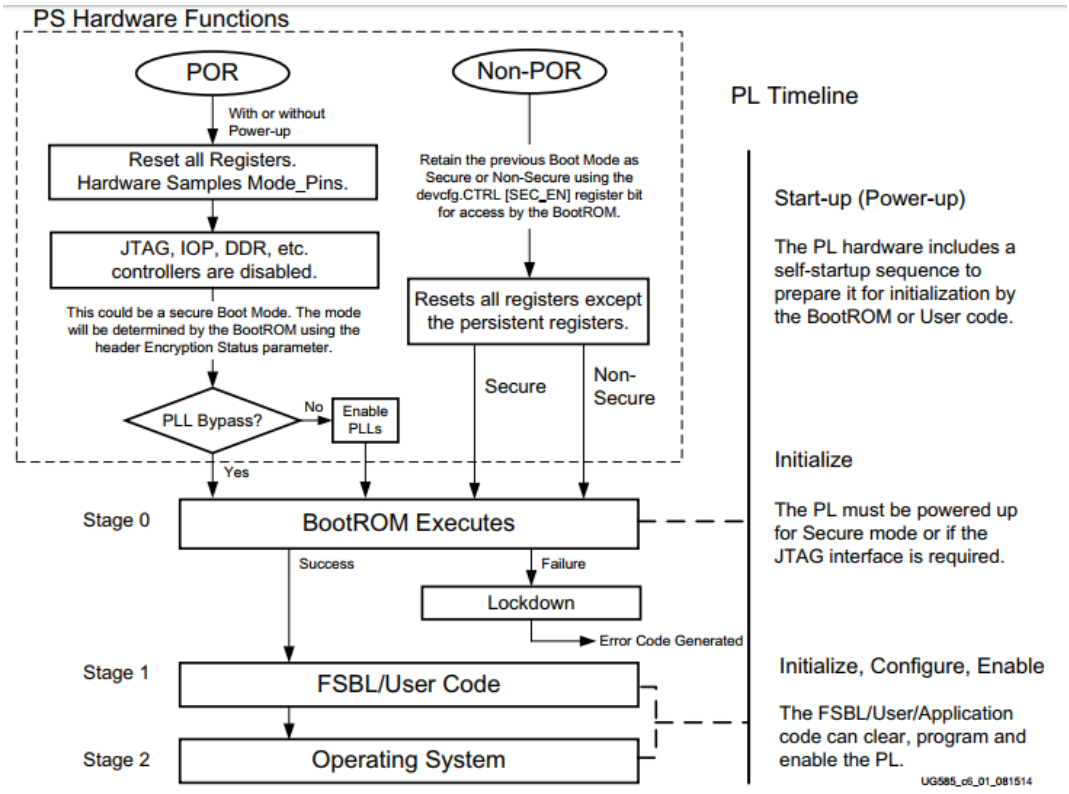
- Konfigurovatelné logické bloky (CLB) - Šesti vstupé look-up tabulky
- Bloková paměť RAM o velikosti 36Kb – Programovatelná FIFO logika
- DSP procesor
- Správu hodin
- Konfigurovatelné vstupně-výstupní zdroje
- Analogově digitální převodníky (XADC)
- PCI Express bloky

2.1.4 Start systému Zynq-7000

Po přivedení napájení na desku ZedBoard a přepnutí přepínače SW8 do stavu ON začne procesorový systém s vykonáváním BootROM kódu. BootROM je první program, který je spuštěn v APU, je vykonáván na všech jádrech a vykoná WFE (wait-for-event/čekání na událost) instrukci. Hlavními úlohami BootROM je konfigurace systému, nakopírování startovací image FSBL nebo uživatelského kódu ze startovacího zařízení do paměti na čipu (256KB RAM OCM) a poté předání vykonávání kódu jednotce OCM [10].

Startovací zařízení prp PS Master Boot Mode může obsahovat jeden nebo více startovacích obrazů. Startovací obraz je vytvořen z hlavičky BootROM a first stage boot loader (FSBL). Startovací zařízení může rovněž obsahovat i bitstream pro konfiguraci programovací logiky a vestavěného operačního systému, tyto soubory ale nejsou přístupné kódu prováděným v BootROM. Pamětí flash pro startování za pomoci PS Master Boot módu

může být například Quad-SPI, NAND, NOR nebo SD karta. V této diplomové práci se zabývám pouze startováním pomocí SD karty [10].



Obrázek 2.5: Diagram startovacího procesu SoC Zynq 7000 [10].

PS Master Boot Mód

V tomto módu probíhá startování za pomoci flash paměti, zde BootROM konfiguruje procesorový systém tak, aby měl přístup k startovacímu zařízení. BootROM dále přečte startovací hlavičku, ověří tuto hlavičku a obvykle zkopíruje FSBL nebo uživatelský kód do paměti na čipu. Pomocí tohoto módu lze použít zabezpečený start nebo nezabezpečený start [10].

Při použití zabezpečeného startu je startovací obraz vždy zapsán do paměti na čipu procesorem. Z paměti na čipu je poté poslán pomocí DMA do a z AES/HMAC jednotek, které slouží k dešifrování a autentizaci. Dešifrovaný startovací obraz je opět zapsán do paměti na čipu. Dešifrovaný startovací obraz se nyní začne vykonávat [10].

Při použití nezabezpečeného startu může BootROM hlavička instruovat procesorový systém k vykonání startovacího obrazu přímo z Quad-SPI nebo NOR startovacího zařízení, v ostatních případech je FSBL nebo uživatelský kód zkopírován do paměti na čipu, kde je vykonán [10].

Pokud je BootROM hlavička na paměťovém médiu invalidní, BootROM hledá další hlavičku. Hledání končí, pokud je nalezena validní hlavička nebo vyhledávání skončilo. Při použití startování z SD karty je povolena pouze jedna hlavička [10].

JTAG Slave Boot

Při použití tohoto módu BootROM vykoná pouze minimální systémovou konfiguraci a povolí JTAG rozhraní. Poté systém přejde do idle stavu čekající na řadič DAP (Debug Access Port), dokud nerestartuje CPU0. Kaskádový JTAG startovací mód cyklí mezi DAP a TAP řadiči. Nezávislý JTAG startovací mód připojí řadič TAP k pinům JTAG programovací logiky a dá uživateli prostor pro konfiguraci programovací logiky pomocí TAP řadiče, který připojí DAP řadič k EMIO JTAG rozhraní [10].

2.1.5 Startovací fáze programovací logiky

Programovací logika musí být napájena před inicializací a nakonfigurování bitstreamem. Spuštění a inicializace programovací logiky se vykonávají nezávisle na procesorovém systému. Programovací logiky musí udržet konkrétní časový vztah s resetovacím signálem POR procesorového systému. Programovací logika může být ovládána pomocí FSBL nebo uživatelským kódem pomocí GPIO pinů nebo za pomoci sériového rozhraní a ostatních externích zařízení. BootROM a FSBL může zjistit stav napájení programovací logiky a získávat přerušení, pokud se stav napájení změní [10].

Startovací průběh programovací logiky má čtyři fáze: nastartování, inicializace, konfigurace a povolení. Startovací fáze je držena, dokud není na programovací logice požadovaná napěťová úroveň. Při inicializační fázi se vyčistí obsah paměti SRAM v programovací logice a připraví ji pro naprogramování pomocí bitstreamu. V konfigurační části probíhá samotné nakonfigurování programovací logiky pomocí bitstreamu. V povolovací fázi je povoleno rozhraní mezi procesorovým systémem a programovací logikou [10].

Hardwarové startovací fáze procesorového systému

Hardwarové startovací fáze procesorového systém zahrnují zajištění napájecího napětí, časování, resetování, vzorkování pinů a inicializaci PLL (fázový závěs). Programovací logika může být spuštěna, zatímco procesorový systém startuje. Během několika hodinových cyklů referenčních hodin (PS_CLK) jsou vzorkovány piny sloužící ke konfiguraci startovacího módu. Jejich hodnoty jsou uloženy do read-only registru. Poté se nainicializuje PLL, pokud je zapnut pomocí propojení pinů JP6 [10].

Softwarové startovací fáze procesorového systému

Startovací fáze se skládá ze tří podfází. Tento proces je ovládán pomocí programu v BootROM a FSBL. Operace BootROM jsou ovlivněny nastavením pinů na desce ZedBoard, hlavičkou BootROM a na základě detekovaných zařízení v systému [10].

Fáze 0 - BootROM/BootROM Header

Nejprve se spustí kód BootROM na prvním jádru. BootROM přečte hlavičku BootROM naprogramovanou na flash zařízení a rozhodne o tom, jak bude start dále probíhat a přesune se do následující fáze. Po hardwarové startovací fázi začnou oba procesory vykonávat stejný BootROM program, který se nachází na adrese 0x0 (počáteční adresa paměti na čipu) kvůli detekci jejich identit. Poté první jádro pokračuje dále vykonáváním BootROM, zatímco druhé jádro vyvoláním instrukce WFE čeká na událost [10].

Fáze 1 - FSBL nebo uživatelský kód

FSBL nebo uživatelský kód je vykonáván hned po první fázi (BootROM). FSBL nebo uživatelský kód rekonfiguruje procesorový systém tak, jak je potřeba a je-li k dispozici bitstream, proběhne nakonfigurování programovací logiky FPGA. Tento kód je vykonáván z paměti na čipu [10].

FSBL vykonává následující operace [10]:

- Inicializace procesorového systému za pomocí PS7 Init dat, která jsou generována vývojovým prostředím Vivado (MIO, DDR atd.).
- Konfigurace programovací logiky pomocí bitstreamu.
- Nahrání U-bootu nebo aplikace do hlavní paměti DDR.
- Předání řízení U-Bootu nebo hlavní aplikaci.

First stage boot loader je plně pod podporou uživatele

Fáze 2 - U-Boot, System, Aplikace

V této fázi může dojít přímo ke startu operačního systému nebo aplikace ale také ke spuštění dalšího bootloaderu například U-boot.

2.1.6 Probuzení druhého jádra

Pro spuštění kódu na druhém jádře je třeba využít první jádro. BootROM dá druhé jádro do stavu, kdy čeká na událost. V tomto stavu není nic povoleno a pouze pár GP registrů bylo modifikováno do stavu, který odpovídá čekání na instrukci [10].

Pokud druhé jádro dostane nějakou systémovou událost, začne okamžitě číst hodnotu adresy 0xFFFFFFF0 a skočí na tuto adresu. Pokud je instrukce SEV (broadcast události všem jádrům) vykonána před změnou cílové adresové lokace 0xFFFFFFF0, druhé jádro pokračuje dále ve stavu čekání na událost, jelikož výchozím obsahem adresy 0xFFFFFFF0 je právě instrukce WFE kvůli bezpečnosti. Pokud je aplikace, která je zapsána na adresu 0xFFFFFFF0 nevalidní nebo ukazuje na nenainicializovanou paměť, výsledek tohoto startu je nepředvídatelný [10].

Pouze ARM-32 ISA kód je podporován pro počáteční skok na druhé jádro. Thumb a Thumb-II instrukce nejsou jako cílová destinace kroku podporovány. Cílová adresa skoku musí být 32 bitová, zarovnaná a musí být validní instrukcí typu ARM-32. Pokud tyto podmínky nejsou splněny, výsledek startování je nepředvídatelný [10].

Pro spuštění aplikace na druhém jádře za pomoci prvního jádra je potřeba:

- Zapsat adresu aplikace pro druhé jádro na adresu 0xFFFFFFF0.
- Po zapsání této adresy vykonat instrukci SEV, po které se druhé jádro probudí a skočí právě na adresu 0xFFFFFFF0, kde je uložena aplikace k provedení.

Při zapisování aplikace na tuto adresu je nutné dbát zvýšené pozornosti, jelikož adresový prostor 0xFFFFFE00 až 0xFFFFFFF0 je rezervován a není dostupný, dokud není fáze 1 plně funkční nebo neběží-li již výše zmíněná aplikace. Jakýkoliv přístup do této oblasti před zapnutím druhého jádra vede k nepředvídatelnému stavu [10].

2.1.7 Obsah startovacího média

Startovací médium může obsahovat několik komponent v několika verzích například [10]:

- BootROM Hlavička – Požadovaná pro BootROM
- FSBL nebo Aplikace ve formátu elf – Požadované pro BootROM, Generován pomocí vývojového prostředí XSDK
- Bitstream programovací logiky – Generováno pomocí vývojového prostředí Vivado
- Aplikace ve formátu elf nebo operační systém – Zkompilovaný cílový program
- Linuxové jádro – V této diplomové práci je zkompilováno pomocí nástroje PetaLinux
- Ramdisk – Není použit v této diplomové práci

2.2 FreeRTOS

Mikrokontrolér je malý procesor s omezenými zdroji, který obsahuje procesor, paměť jen pro čtení (ROM nebo Flash) obsahující program, který bude vykonáván a paměť RAM (Random Access Memory) potřebnou pro aplikace, které procesor vykonává. Typicky je program vykonáván přímo z ROM (Read Only Memory). Mikrokontroléry jsou používány ve vestavěných systémech, které vykonávají specifickou úlohu. FreeRTOS je třída RTOS systému, který je vyvinut tak, aby byl dostatečně malý pro spuštění na mikrokontrolérech. FreeRTOS je přenesen na mnoho architektur. Pro tuto diplomovou práci je použit port pro vývojový kit ZC702, což je vývojová deska od firmy Xilinx podobná ZedBoard. Obě desky používají SoC Zynq-7000 [13].

FreeRTOS poskytuje základní Real-Time plánovací funkcionalitu, inter-task komunikaci, časování a synchronizaci [13].

2.2.1 Úlohy

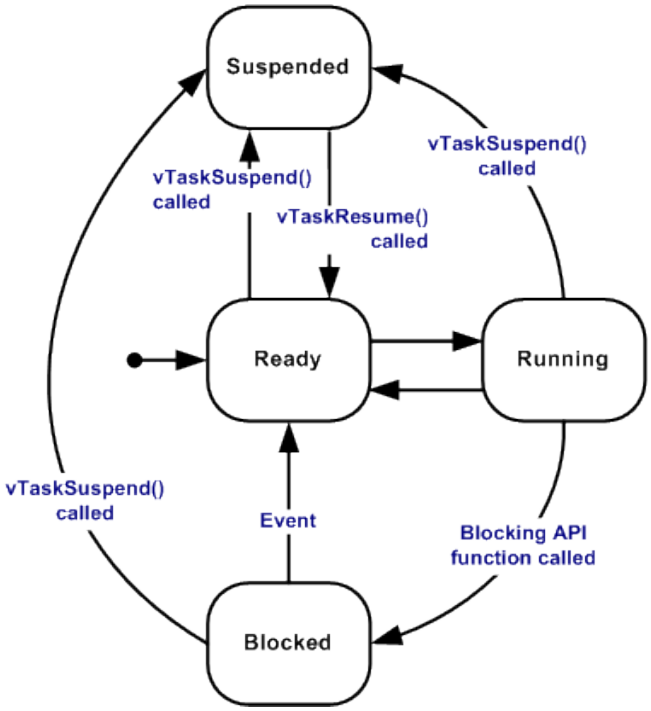
Real-Time aplikace, která využívá RTOS může být strukturována jako kolekce nezávislých úloh. Každá úloha je vykonávána v rámci svého kontextu nezávisle na ostatních úlohách nebo na RTOS plánovači samotném. Úloha nemá žádné znalosti o aktivitě RTOS plánovače. Povinností Real-Time RTOS plánovače je zajistit, aby kontext na procesoru byl totožný s kontextem právě vykonávané úlohy. K dosažení tohoto cíle má každá úloha k dispozici svůj vlastní zásobník. Pokud je úloha vyjmuta z vykonávaného kontextu, je stav úlohy uložen na lokální zásobník úlohy. Díky uložení stavu úlohy lze při opětovném vykonání úlohy pokračovat na přerušeném místě [13].

Mezi výhody FreeRTOS úlohy patří jednoduchost implementace, žádné omezení na použití úlohy, podpora preemptivnosti a možnost mít přidělenou prioritu. Mezi nevýhody se řadí zejména vyšší použití paměti RAM (každá úloha má svůj zásobník) [13].

Stavy úlohy ve FreeRTOS

Úloha může existovat pouze v jednom z níže uvedených stavů [13]:

- Running (Spuštěna) - Úloha, která je aktuálně spuštěna a je jí přidělen procesorový čas. V jednojádrovém procesoru může být spuštěna pouze jedna úloha.
- Ready (Připravena) - Úlohy, které jsou připraveny k běhu, ale neběží, protože procesor je přidělen úloze s vyšší nebo stejnou prioritou.
- Blocked (Blokována) - Úloha je blokována, pokud aktuálně čeká na nějakou událost. Například zavolá-li úloha funkci vTaskDelay, úloha je blokována na určitou periodu.
- Suspended (Suspendována) - Úloha je suspendována pouze při explicitním zavolání funkcí vTaskSuspend(). Úloha ve Suspended stavu nemůže být nikdy spuštěna, ale lze ji zpět zařadit do fronty připravených úloh k běhu pomocí funkce xTaskResume().



Obrázek 2.6: Validní přechody mezi stavy úloh [13].

Priority úloh

Každá úloha má přiřazenou prioritu od 0 do *(configMAX_PRIORITIES - 1)*, kde *configMAX_PRIORITIES* je definován v hlavičkovém souboru freeRTOSConfig.h. Malá prioritní čísla značí nízko prioritní úlohy. Úloha idle je spuštěna vždy při spuštění systému a má nejmenší možnou prioritu v systému tj. 0. Úloha idle se stará o uvolňování paměti alokované RTOS systémem úlohám, které již byly odstraněny pomocí *vTaskDelete()*. Maximální priorita může být na 32 bitovém systému rovna 0xFFFFFFFF (4294967295). Maximální prioritu je vhodné volit s ohledem na systémové prostředky [13].

Implementace úloh

Typ *TaskFunction_t* je definován jako funkce, která vrací void a jejím argumentem je ukazatel na typ void. Všechny funkce, které implementují úlohy, by měly být tohoto typu. Argument lze použít pro předávání informací jakéhokoliv typu. Funkce úloh by nikdy neměly nic vracet, jelikož jsou typicky implementovány jako nekonečný cyklus [13].

Úlohy jsou vytvořeny zavoláním *xTaskCreate()* a *xTaskCreateStatic()*. Smazány jsou pomocí *vTaskDelete()* [13].

```
void vATaskFunction( void *pvParameters ) {
    for( ;; ) {
        // Nekonecny cyklus aplikace
    }
    vTaskDelete( NULL );
}
```

2.2.2 Synchronizace procesů a komunikace mezi procesy

Fronty

Fronty jsou ve FreeRTOS určeny primárně pro meziúlohovou komunikaci. Mohou být použity k zasílání zpráv mezi úlohami ale také i mezi přerušeními a úlohami. Ve většině případů jsou použity jako vzhledem k jádru bezpečné FIFO buffery, kde se nová data řadí na konec fronty (*xQueueSendToBack()*). Data mohou být přečtená pomocí funkce *xQueueReceive()* [13].

Binární semafory

Binární semafory jsou nejen využity pro výlučný přístup, ale i pro synchronizaci procesů. Mutexy a binární semafory jsou si hodně podobné, největším rozdílem je absence mechanismu dědění priority. Proto jsou binární semafory vhodnější spíše na synchronizaci a mutexy na implementaci jednoduchého výlučného přístupu [13].

Rozhraní pro práci se semafory dovoluje specifikovat blokovací čas (block time). Blokovací čas indikuje maximální počet tiků hodin, po který by měla úloha vstoupit do blokovaného stavu při pokusu o získání semaforu. Semafor nemusí být k dispozici okamžitě. Pokud se jedna nebo více úloh navzájem blokují na stejném semaforu, je odblokována úloha s vyšší prioritou [13].

Binární semafory s čítačem

Binární semafory s čítačem dovolují vstup více procesům do kritické sekce. Mají definovanou kapacitu, která značí kolik procesů může současně semafor získat [13].

Typicky se využívají pro [13]:

- Čítání událostí – Inicializace semaforu na 0. Při výskytu události se čítač semaforu inkrementuje. Hodnota čítače tedy signalizuje počet událostí připravených ke zpracování.
- Správu zdrojů – Inicializace čítače semaforu na maximální počet procesů, které mohou současně přistupovat ke sdílenému zdroji. Při každém získání sdíleného zdroje je čítač

dekrementován. Je-li čítač na hodnotě 0, žádný proces jíž nemůže získat sdílený zdroj a musí čekat na uvolnění semaforu některým procesem, v jehož držení sdílený zdroj je.

2.2.3 Mutexy

Mutexy implementují mechanismus dědění priorit, využívají se zejména pro výlučný přístup. Mutex se chová jako token, pokud je využitý jako prostředek pro zajištění výlučného přístupu, který je využit pro ochranu zdroje. Pokud některá z úloh potřebuje získat zdroj, musí nejprve získat token. Po získání tokenu může úloha pracovat se sdíleným zdrojem. Má-li některá úloha token, jiné úlohy nemohou získat zdroj. Mutex může být odemčen pouze vlastníkem tokenu. Po ukončení práce se zdroji vrátí úloha token mutexu a další úlohy mohou získat sdílený zdroj [13].

Mutexy používají stejné rozhraní (API) jako binární semafory s tím rozdílem, že implementují mechanismus dědění priorit. To znamená, že pokud je zdroj získán nízko prioritní úlohou a vysoko prioritní úloha je blokována při získávaní zdroje, je priorita úlohy, která vlastní token, zvýšena na úroveň priority vysoko prioritního procesu. Tím je minimalizován jev nazvaný inverze priorit, ke kterému by docházelo.

Mutexy by neměly být používány při přerušení ze dvou důvodů [13]:

- Dědění priorit dává smysl pouze, je-li token přidělen/získán od úloh, ne od přerušení.
- Přerušení nemůže být blokováno kvůli čekání na zdroj, který je chráněn mutexem.

2.2.4 Správa paměti

Jádro RTOS potřebuje RAM pokaždé, když je vytvořena úloha, fronta, mutex, softwarový čítač, semafor nebo skupina událostí. Paměť RAM může být dynamicky alokována z RTOS hromady v rámci RTOS API funkcí vytváření projektů, nebo může být napsána uživatelem. FreeRTOS zachovává API alokaci paměti v přenositelné vrstvě. Přenositelná vrstva (Portable layer) je implementována mimo základní RTOS funkcionalitu, tím FreeRTOS dovoluje použití vhodné implementace alokování paměti. Pokud jádro RTOS požaduje alokaci paměťi, místo volání funkce malloc() je zavolána funkce pvPortMalloc(). Podobně při dealokaci je zavolána místo funkce free() funkce vPortFree(). FreeRTOS poskytuje několik správ hromady, každá varianta se liší v komplexnosti a vlastnostech. Je dokonce možné použít dvě varianty zároveň [13].

Varianty správy hromady jsou popsány níže [13]:

- heap_1 - Nejjednodušší, nepovoluje uvolňování paměti.
- heap_2 - Povoluje uvolňování paměti, ale neumožňuje sdružování sousedních volných bloků.
- heap_3 - Zabalí funkce malloc() a free() tak, aby byly bezpečné pro jádro.
- heap_4 - Sdružuje sousední volné bloky a zabraňuje tím fragmentaci. Zahrnuje volbu umístění absolutních adres.
- heap_5 - Stejná funkcionalita jako heap_4, s tím že dokáže rozšířit hromadu přes několik nesouvislých paměťových oblastí.

Kapitola 3

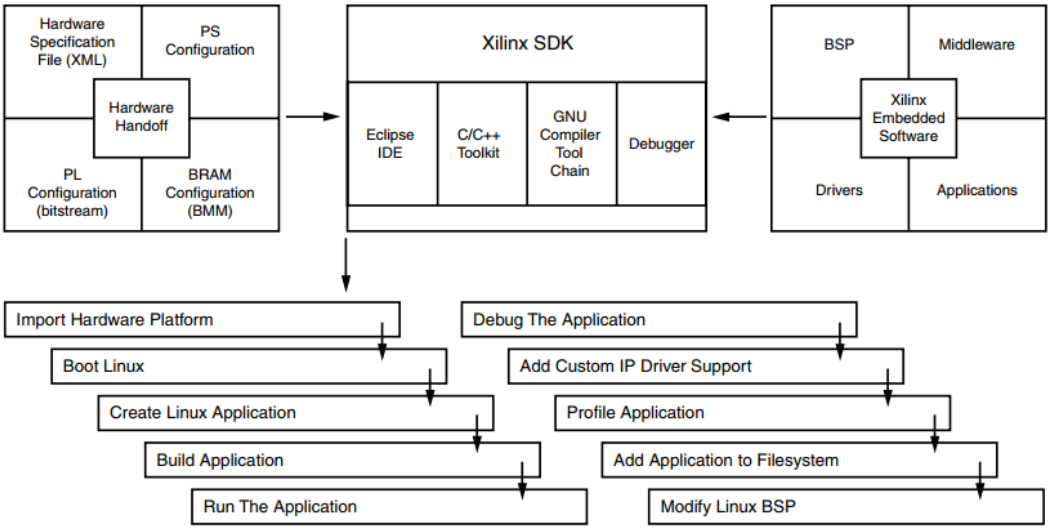
Realizace

3.1 Realizace asymetrického multiprocesorového systému – AMP

Tato kapitola pojednává o asymetrickém multiprocesorovém systému AMP, ve kterém na každém jádru ve vícejádrovém systému běží různé operační systémy a sdílí mezi sebou paměť. Tyto operační systémy mohou být stejné, ale typicky je každý operační systém jiný z důvodu doplnění ostatních operačních systémů o základní funkce, například spuštění operačního systému Linux na prvním jádru a na druhém jádru například FreeRTOS. První jádro poskytuje všechny základní funkce, jako komunikace po síti a propojení externích zařízení se systémem, zatímco na druhém jádře běží real time aplikace. Kritickým prvkem tohoto přístupu je rozdělení systémových zařízení (ethernet, UART). Většina systémových zařízení musí být přiřazena právě jednomu jádru. Řadič přerušení je navržen pro jeho sdílení mezi více jádry, ale pouze jedno z jader je master. Master jádro inicializuje řadič přerušení. Komunikace mezi těmito jádry je pro tento přístup klíčová, lze jej řešit například sdílenou pamětí, předáváním zpráv a přerušeními mezi procesory [9].

3.1.1 Hardwarová část

Přizpůsobení hardwarové části SoC Zynq-7000 spočívá ve více krocích. Nejprve je nutné za pomocí vývojového prostředí Vivado vytvořit blokový diagram procesorového systému Zynq. Přidat a nakonfigurovat k němu další IP bloky, tyto bloky propojit a nadefinovat vstupně-výstupní porty. Dále vytvořit k navrženému hardwarovému řešení top level VHDL nebo Verilog soubor. Nakonec tento návrh vysyntetizovat, implementovat a vygenerovat z tohoto návrh soubor bitstream. Po splnění těchto kroků je nutné exportovat hardwarovou specifikaci tohoto návrhu do nástroje XSDK. Při exportování budou vytvořeny a vyexportovány další čtyři soubory a to ps7_init.c, ps7_init.h, ps7_init.tcl, a ps7_init.html.



Obrázek 3.1: Vývojový diagram aplikací na platformu Zynq-7000 [9]

Hardware Platform Specifikace

Hardware Platform Specification obsahuje všechny informace o vytvořeném hardwarovém návrhu. Soubor ve formátu XML obsahuje souhrnné informace o adresové mapě navrženého hardwarového designu a seznam použitých IP bloků.

Address Map for processor ps7_cortexa9_[0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
Sws_8bits	0x41200000	0x4120ffff		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
BTNs_5bits	0x41240000	0x4124ffff		REGISTER

Obrázek 3.2: Adresová mapa návrhu

IP blocks present in the design

ps7_intc_dist_0	ps7_intc_dist	1.00.a
ps7_gpio_0	ps7_gpio	1.00.a
Sws_8bits	axi_gpio	2.0
ps7_scutimer_0	ps7_scutimer	1.00.a
ps7_slcr_0	ps7_slcr	1.00.a
ps7_scuwdt_0	ps7_scuwdt	1.00.a
ps7_l2cachec_0	ps7_l2cachec	1.00.a
ps7_scuc_0	ps7_scuc	1.00.a
ps7_qspi_linear_0	ps7_qspi_linear	1.00.a
BTNs_5bits	axi_gpio	2.0

Obrázek 3.3: Seznam použitých IP bloků

Inicializační soubory ps7*

Tyto soubory obsahují inicializační kód pro procesorový systém čipu Zynq a inicializační nastavení pro DDR paměti, hodiny, pll obvody a MIO. Vývojové prostředí XSDK tyto soubory používá k inicializaci procesorového systému tak, aby budoucí vytvořené aplikace mohly používat všechny periferie navrženého systému.

Xilinx Vivado Design Suite

Vivado Design Suite je vývojové prostředí pro vytváření FPGA návrhů, jeho autorem je společnost Xilinx, která jej nabízí na svých stránkách zdarma pro Linux, Windows XP a Windows 7 a novější. Toto vývojové prostředí plně podporuje právě zvolenou platformu ZedBoard. Vivado poskytuje základní nástroj pro vytvoření FPGA návrhů v RTL, syntézy, implementaci a verifikaci. Obsahuje nástroj IP Integrator, který může být použit pro návrh hardwarové části vestavěného procesorového systému. Pomocí tohoto nástroje lze přidávat k návrhu například další IP ARM Cortex A9 jádra, IP periferie a vytvářet komunikaci mezi nimi.

3.1.2 Změny v hardwarovém návrhu

Pro běh operačního systému Linux na prvním jádře a FreeRTOS na druhém jádře je zapotřebí upravit výchozí konfiguraci procesorového systému Zynq-7000.

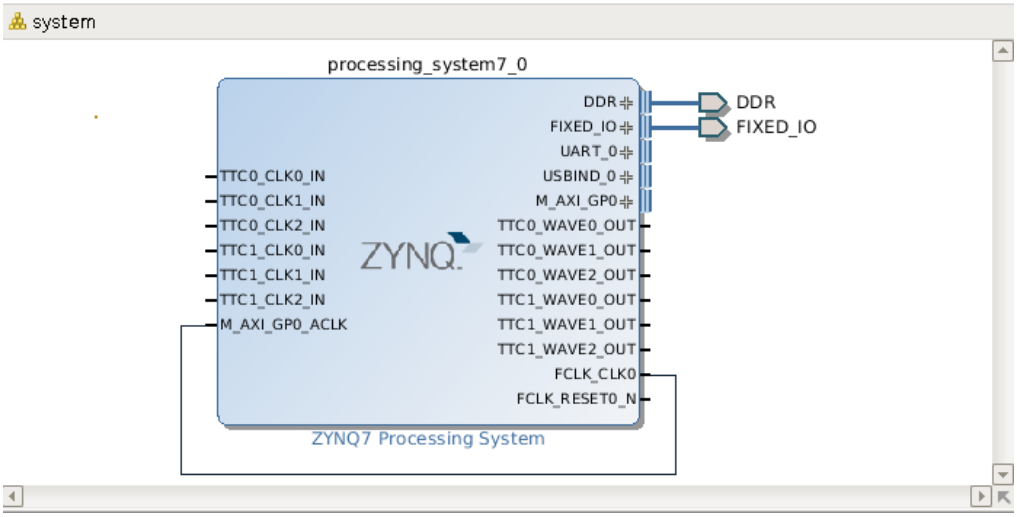
Operační systém Linux vyžaduje [47]:

- Jeden TTC (Triple Timer Counter)
- Externí paměť o velikosti 32 MB a více (DDR)
- UART pro sériovou komunikaci
- Ethernet pro komunikaci po síti (volitelné)
- Nevolatilní paměť (volitelné)

Operační systém FreeRTOS vyžaduje:

- Jeden TTC (Triple Timer Counter)
- UART pro sériovou komunikaci

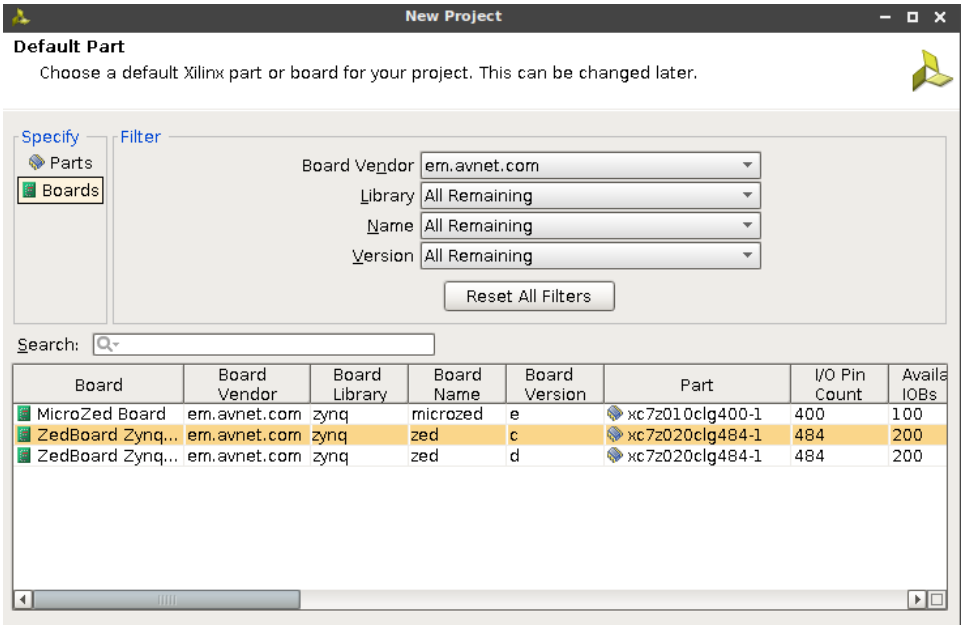
Oba operační systémy vyžadují jeden čítač a UART rozhraní, tyto periferie jsou ale v základní konfiguraci Zynq-7000 zastoupeny pouze jednou. Proto je nutné upravit tuto konfiguraci tak, aby rozhraní UART1 využívalo jádro, na kterém běží operační systém Linux a UART0 pro jádro na kterém běží FreeRTOS. Dále je nutné povolit další časovač (Timer1). Po jejich přidání je nutné nakonfigurovat jejich vstupy a výstupy jako EMIO. Dále je možné k tomuto návrhu přidat IP bloky pro ovládání LED diod, přepínačů a tlačítek. Tyto jednotlivé IP Bloky mohou být připojeny k procesorovému systém za pomocí AXI interconnectu. Tato změna však z hlediska AMP systému není nutná.



Obrázek 3.4: Výsledný blokový diagram hardwarového návrhu

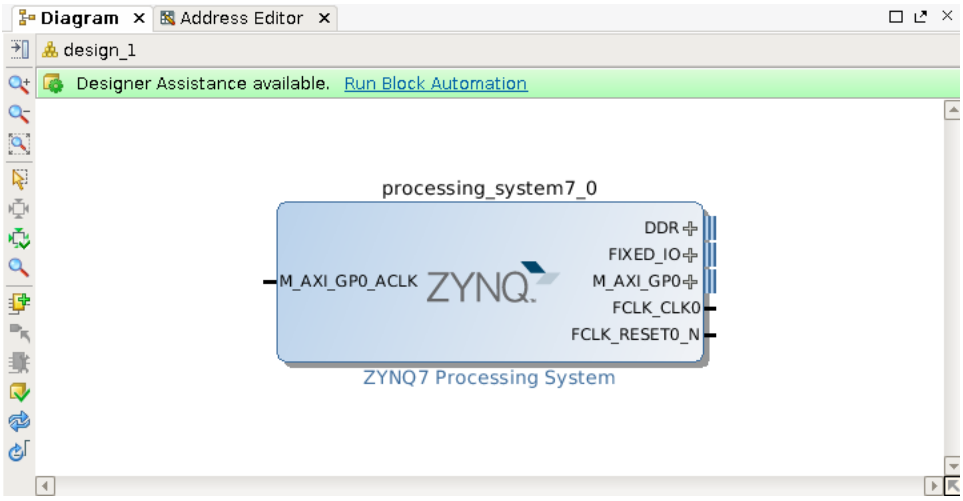
3.1.3 Realizace změn

Nejprve je nutné pomocí vývojového prostředí Vivado vytvořit nový projekt typu RTL (RTL Project). Jako defaultní desku pro tento projekt je důležité vybrat desku ZedBoard revize c s označením xc7z020clg484-1. Právě tato vývojová deska je vybrána jako realizační prostředek této diplomové práce. Obrazovka výběru desky by měla vypadat následovně:



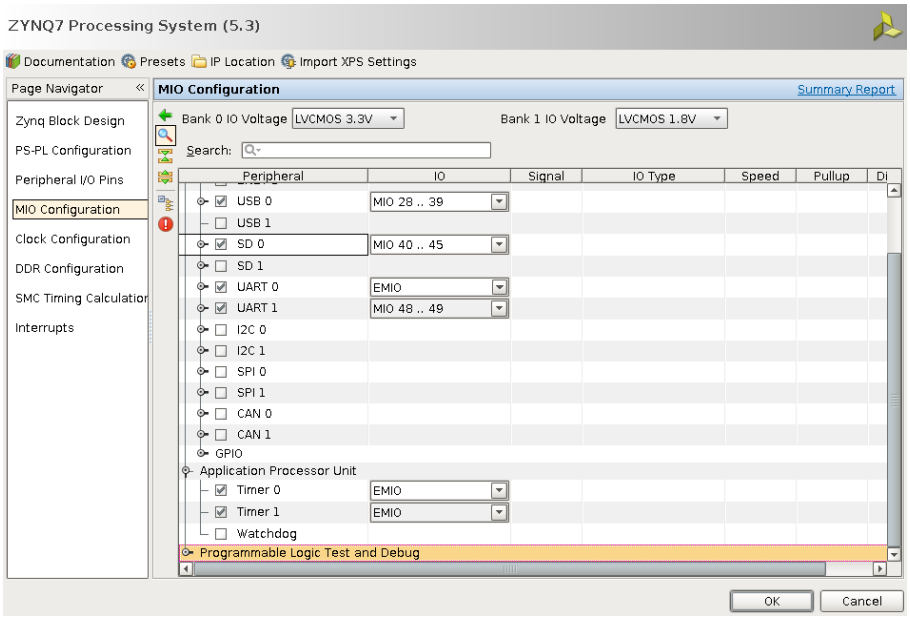
Obrázek 3.5: Výběr ZedBoard platformy ve vývojovém prostředí Vivado

Po vytvoření projektu vzniknul prvotní projekt. Nyní je nutné vytvořit blokový design pomocí vnitřně integrovaného IP Integratoru. Tento design bude po inicializaci prázdný, je proto nutné do něj přidat IP blok SoC desky ZedBoard, tj. Zynq7 Processing System. Design by měl prozatím obsahovat block procesorového systému Zynq 7000.



Obrázek 3.6: Blokový diagram obsahující procesorový systém Zynq-7000

Po dvojitém kliknutí na tento IP blok se zobrazí okno, které umožňuje konfigurovat periferie a ostatní nastavení procesorového systému Zynq 7000. Při první konfiguraci je nutné zinicializovat nastavení procesorového systému nastavením uloženého výchozího nastavení (presetu) pro vývojový kit ZedBoard. Pro realizaci výše uvedených změn je nutné změnit nastavení MIO, konkrétně v nastavení vstupně-výstupních periferií povolit UART0 a namapovat jej na EMIO a v aplikační procesorové jednotce povolit další řítač (Timer1) a rovněž jej namapovat na EMIO.



Obrázek 3.7: Nastavení periferií Timer1 a UART0

Po rekonfiguraci je možné spustit asistenta automatické konfigurace blokového designu. Tento asistent připojí IO_FIXED a DDR na výstupní porty. Pro správnou funkci procesorového systému je důležité připojit hodiny, tj. připojení FCLK_CLK0 na M_AXI_GP0_ACLK.

Pro další potřeby je v tomto designu možné přidat rozhraní například pro LED diody a přepínače, ale z hlediska ukázky AMP řešení to není nutné.

V dalším kroku je třeba vytvořit HDL vrstvu pro tento design a vytvořený design vysyntetizovat, implementovat a simulovat (ve Vivadu Generate Output Products). Pro vyexportování bitstreamu ve vývojovém prostředí Vivado je k dispozici tlačítko Generate Bitstrea. Po dokončení všech operací je možné celkový design vyexportovat do vývojového prostředí XSDK.

3.1.4 Softwarová část

V této části je nutné vygenerovat BSP balíčky aplikací a zavaděčů pomocí již dříve vytvořené hardwarové specifikace. BSP balíčky neboli Board Support Packages jsou balíčky pro inicializaci desky ZedBoard a čipu Zynq-7000 při přivedení napájení podle exportované hardwarové specifikace. Tato inicializace vytvoří prostředí pro aplikace založené na tomto BSP. Všechny tyto balíčky jsou vytvořeny pomocí XSDK.

PetaLinux Tools

Nástroje PetaLinux jsou vytvořeny společností Xilinx, která tyto nástroje poskytuje všem zájemcům zdarma. Jsou určeny pro vývojáře k nakonfigurování, sestavení a nasazení systémů na čipy firmy Xilinx. PetaLinux nevytváří Linuxovou distribuci, pouze usnadňuje práci s nástroji, které jsou dostupné na git repositáři firmy Xilinx [46].
PetaLinux dokáže vytvářet [46]:

- FSBL
- U-Boot
- ARM Trusted Firmware
- Linux (jádro)
- Knihovny a aplikace
- Xen Hypervisor

Aplikační balíček FSBL pro AMP

Tento balíček je určen pro první jádro procesorového systému Zynq-7000. Je generován jako standalone, což značí, že balíček nevyužívá žádný operační systém. Tento balíček tedy obsahuje FSBL pro AMP systém, který bude spuštěn na prvním jádru.

Vytvoření BSP balíčku pro FreeRTOS

V tomto balíčku je nutné namapovat ps7_uart_0 na STDOUT a STDIN. Dále je nutné k parametrům překladače přidat -DUSE_AMP=1.

Name	Value	Default	Type	Description
compiler	arm-xilinx-eabi-gcc	arm-xilinx-eabi-gcc	string	Compiler used to compile
archiver	arm-xilinx-eabi-ar	arm-xilinx-eabi-ar	string	Archiver used to archive li
compiler_flags	-O2 -c	-O2 -c	string	Compiler flags used in BSF
extra_compiler_flags	-g -DUSE_AMP=1	-g	string	Extra compiler flags used

Obrázek 3.8: Úprava výchozích flagů překladače

Vytvoření BSP balíčku pro PetaLinux

V tomto balíčku je nutné správně namapovat zařízení, které Linux využívá na fyzické zařízení v čipu.

Nastavení BSP balíčku pro PetaLinux je následující:

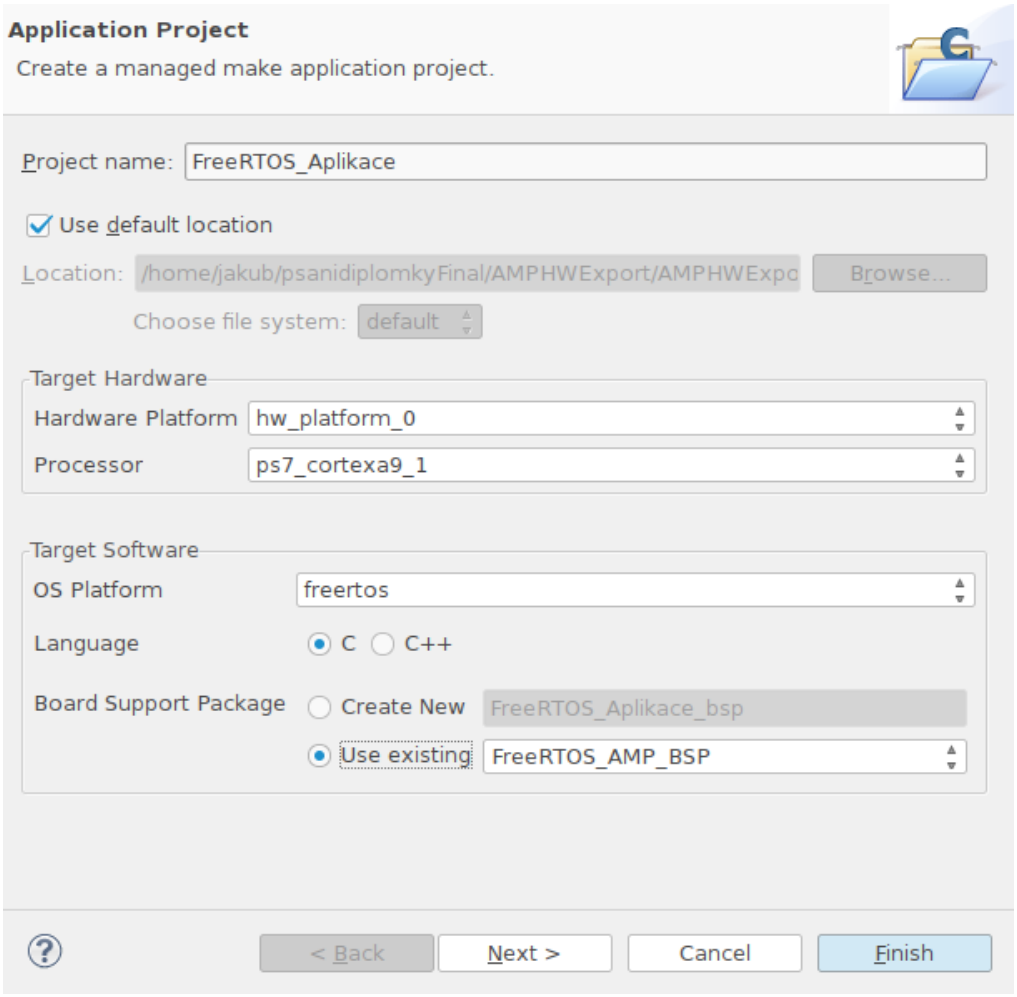
- Namapování ps7_uart_1 na STDOUT a STDIN
- ps7_ethernet na rozhraní Ethernet
- ps7_sd_0 na rozhraní SD
- ps7_ddr_0 na hlavní paměť
- ps7_qspi na paměť flash

TARGET_DIR			string	Destination directory for F
bootargs			string	Booting arguments
ethernet	ps7_ethernet_0	none	peripheral	Ethernet controller
flash_memory	ps7_qspi_0	none	peripheral	Name of Flash Memory Co
flash_memory_bank	0	0		Bank Nr. within memory c
flash_memory_offset	0	0		Manual override of flash m
flash_memory_size	0	0		Manual override of flash m
flash_memory_start	-1	-1		Manual override of flash m
gpio	none	none	peripheral	Led interface
iic	none	none	peripheral	iic interface
lmb_memory	none	none	peripheral	Name of LMB Memory use
main_memory	ps7_ddr_0	none	peripheral	Name of Main Memory use
main_memory_bank	0	0		Bank Nr. within memory c
main_memory_offset	0	0		Manual override of main m
main_memory_size	0	0		Manual override of main m
main_memory_start	-1	-1		Manual override of main m
periph_type_overrides			string	List of peripheral type ove
sdio	ps7_sd_0	none	peripheral	SDIO/SD
stdin	ps7_uart_1	none	peripheral	stdin peripheral for bootlo
stdout	ps7_uart_1	none	peripheral	stdout peripheral for bootl
sysace	none	none	peripheral	Sysace interface for CF
timer	none	none	peripheral	Timer for measuring time
uboot_position	0	0		Uboot position in memory
▶ microblaze_exceptions	false	false	boolean	Setting ignored

Obrázek 3.9: Mapování zařízení na zařízení v čipu

Aplikační balíček FSBL pro FreeRTOS

Aplikační balíček obsahuje zkompilovanou aplikaci ve formátu elf, která bude spuštěna v operačním systému FreeRTOS. Cílem tohoto balíčku je druhé jádro. Jako BSP balíček je nutné použít BSP balíček pro FreeRTOS.



Obrázek 3.10: Nastavení aplikačního projektu pro druhé FreeRTOS

Konfigurace jádra Linux

Při konfiguraci jádra je nutné povolit dynamické načítání modulů, upravit rozdělení paměti a povolit ovladač remoteproc. Dále je třeba do DTS souboru čipu Zynq přidat fragment obsahující potřebné informace o remoteproc ovladači. Tímto fragmentem určíme adresový prostor ovladače remoteproc (od adresy 0x10000000 je již jádro Linux) a nasdílíme tomuto ovladači přerušení. Položka ipino obsahuje číslo meziprocesorového přerušení [48].

```
test: remoteproc-test@0 {
   compatible = "xlnx,zynq_remoteproc";
   reg = < 0x0 0x10000000 >;
   interrupt-parent = <&ps7_scugic_0>;
   interrupts = < 0 37 4 0 38 4>;
   firmware = "freertos";
   ipino = <6>;
   vring0 = <2>;
   vring1 = <3>;
} ;
```

Vygenerování souborů pro startování systému z SD karty

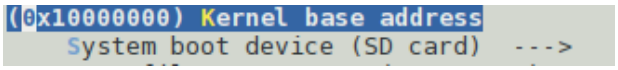
Pro vygenerování souborů je možné využít výše popsaný nástroj PetaLinux. Pro vytvoření nového projektu použijeme nástroj petalinux-create.

```
petalinux-create -t project -n AMPFinalPetaLinux
```

Po vytvoření prázdného projektu je nutné tento projekt konfigurovat tak, aby reflektoval námi navržený hardwarový design, toho docílíme použitím konfiguračního nástroje petalinux-config s přepínačem –get-hw-description. Pro konfiguraci PetaLinuxu použijeme BSP balíček vytvořený pro PetaLinux. Během konfigurace je mimo jiné vytvořen DTS soubor, který slouží k popisu hardware systému. DTS neboli Device Tree Source je během startovací fáze předán jako argument jádru Linux a ten pomocí tohoto popisu nainicializuje požadované periferie.

```
petalinux-config --get-hw-description -p PATH_TO_PROJECT/
INFO: Checking component...
INFO: Getting hardware description...
INFO: Using MSS file PATH_TO_PETALINUX_BSP/PetaLinux_BSP_AMP/system.mss
and XML file PATH_TO_PETALINUX_BSP/PetaLinux_BSP_AMP/../hw_platform_0/system.xml
INFO: Copy autoconfig for PetaLinux project: /PATH_TO_PROJECT
INFO: Merging platform settings into kernel configuration
Auto-config file successfully updated for PetaLinux project: PATH_TO_PROJECT/
[INFO ] generate PATH_TO_PROJECT/subsystems/linux/hw-description/system.dts
```

Poté nakonfigurujume PetaLinux pomocí petalinux-config. V této fázi je důležité nastavit startovací zařízení na SD kartu a bázovou adresu jádra Linux.



Obrázek 3.11: Nastavení PetaLinuxu

Dalším krokem je konfigurace jádra Linux. To je provedeno opět pomocí nástroje petalinux-config s argumentem -c kernel. Obsah souborového systému root je možné uzpůsobit pomocí petalinux-config -c rootfs.

Následně přidáme do PetaLinuxu námi vygenerovaný aplikační projekt pro FreeRTOS.

```
petalinux-create -t apps --enable -n FreeRTOS_Aplikace
INFO: Create apps: FreeRTOS_Aplikace
INFO: New apps successfully created in
   PATH_TO_PROJECT/components/apps/FreeRTOS_Aplikace
cp PATH_TO_FREERTOS_APP/FreeRTOS_Aplikace/Debug/FreeRTOS_Aplikace.elf
   components/apps/FreeRTOS_Aplikace/data/FreeRTOS_Aplikace
```

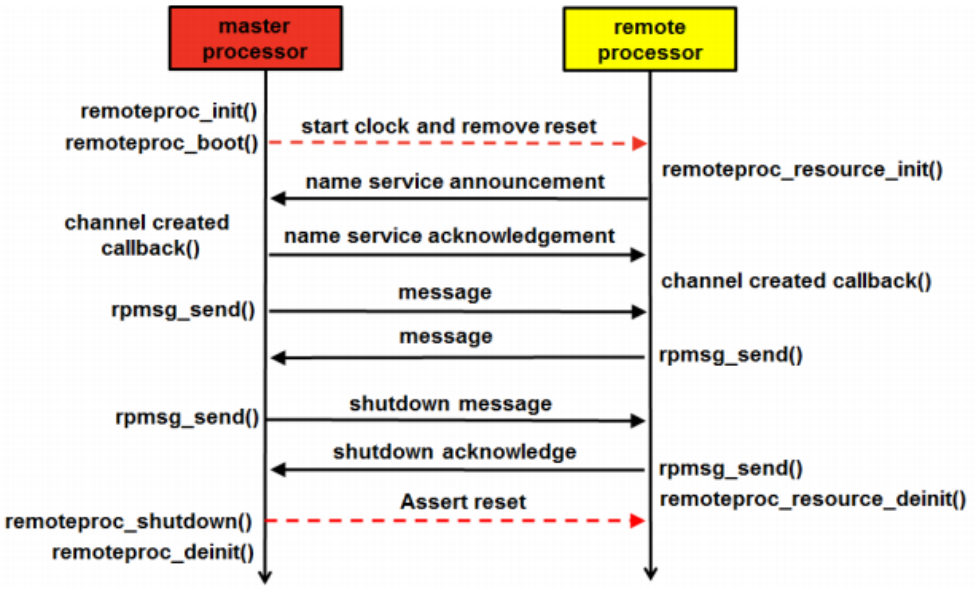
V posledním kroku vše sestavíme a připravíme startovací soubory BOOT.bin a image.ub, které zkopírujeme na SD kartu a pomocí této SD karty přivedeme napětí do ZedBoardu.

```
petalinux-build
petalinux-package --boot --fsbl ELF_OF_FSBL
  --fpga BITSTREAM --uboot --force -o OUTPUT_DIR/BOOT.BIN
petalinux-package --prebuilt --fpga BITSTREAM --force
```

3.1.5 Komunikace mezi jádry

Při Linux-FreeRTOS konfiguraci se dá využít možnosti nastavení druhého jádra pro FreeRTOS a nahrání firmware FreeRTOS na druhé jádro. Ovladač remoteproc ovládá proces vkládání a odebrání AMP modulů na druhém jádře. Umožňuje tedy odepnout druhé jádro od operačního systému Linux, nakonfigurovat jej a nahrát FreeRTOS firmware do paměťové oblasti druhého jádra. Ovladače remoteproc jsou součástí ovladačů jádra Linux stejně jako součástí FreeRTOS knihoven, které ovládají a spravují paměť a přerušení mezi procesorové komunikace. Pomocí tohoto ovladače je umožněno rozhraní rpmsg zasílat zprávy mezi Linuxem a FreeRTOS. Dále je tento ovladač odpovědný za sdílení hardwarových periférií mezi jádry a dovoluje jádrům procesoru přistupovat do sdílené paměti [48].

Pro zasílání a přijímání zpráv je použito rozhraní rpmsg, které využívá tzv. VRING buffery. Tyto buffery obsahují zprávy poslané jádrem, na kterém běží FreeRTOS. Jsou-li zprávy vloženy do VRING bufferu, odesílatel oznámí přijímacímu procesoru novou zprávu za pomocí přerušení. Trace Buffer je předalokovaný segment paměti pro FreeRTOS systém, využívaný jako logovací buffer. Tento buffer pouze zpřístupňuje logovací zprávy FreeRTOS systému bez použití UART0 rozhraní [48].



Obrázek 3.12: Komunikace mezi jádry pomocí remoteproc ovladače [48]

3.1.6 Změny v operačním systému

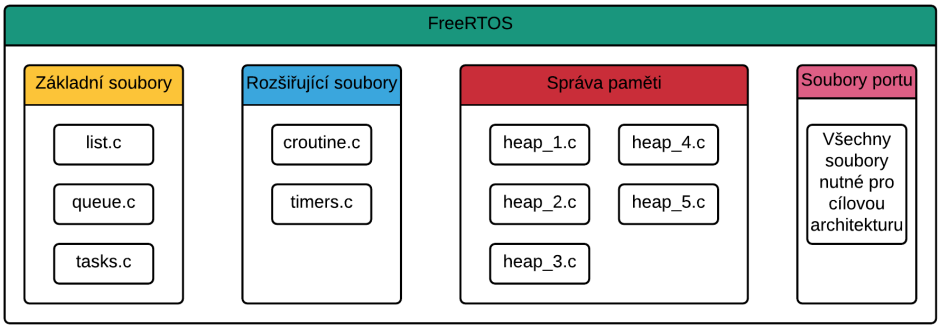
V případě použití AMP konfigurace není nutné z hlediska operačního systému provádět žádné změny. Každý procesor má svou L1 vyrovnávací paměť a pro zjednodušení lze sdílenou vyrovnávací paměť úrovně L2 vypnout. V rámci současného běhu jiných instancí operačního systému nedochází k plánování úloh mezi jádry, úlohy si nepotřebují být vědomy na jakém procesoru běží a každý operační systém má vyhrazenou paměť (například ZedBoard obsahuje 512 MB, což je rozdělení 256 MB pro první jádro a 256 MB pro druhé jádro).

3.2 Realizace symetrického multiprocesorového systému – SMP

Z hlediska hardwarového návrhu je nutné pouze splnit dostatek zdrojů pro požadovaný operační systém, například pro FreeRTOS to je UART rozhraní a časovač.

3.2.1 Softwarová část

Základní kód operačního systému FreeRTOS je rozdělen do tří souborů nazvaných tasks.c, queue.c a list.c. Soubor tasks.c obsahuje kód realizující správu úloh. Soubor list.c implementuje datovou strukturu typu seznam, která je používána plánovačem a soubor queue.c poskytuje datovou strukturu typu fronta, která je využívána při předávání zpráv. Dále FreeRTOS implementuje softwarové časovače v souboru timers.c a co-routiny (funkce, které jsou efektivní z hlediska paměti) v souboru croutine.c. Jednotlivé typy implementací správy paměti jsou umístěny v souborech pojmenovaných jako heap_1 až 5. Dále obsahuje soubory nutné pro korektní kompilaci a běh FreeRTOS na cílové architektuře. FreeRTOS nyní podporuje více než 25 architektur. Pro ZedBoard lze využít port pro ZC702 a port pro kompilátor ARM_CA9.



Obrázek 3.13: Struktura FreeRTOS systému

Změny v operačním systému

Změny musí být provedeny v plánování a synchronizaci, z hlediska atomických operací nejsou třeba žádné změny, protože jádra ARM, již obsahují atomické instrukce pracující na všech jádrech. Pro alokaci a dealokaci je použita třetí implementace správy paměti FreeRTOS (heap_3.c). Druhé jádro musí být zapnuto za pomocí prvního jádra viz. 2.1.6.

Plánovač ve FreeRTOS

FreeRTOS využívá preemptivní plánovač, jenž plánuje úlohy na základě jejich statické priority. Při vybírání procesu vybere proces s nejvyšší prioritou, po vybrání procesu je tato událost oznámena procesoru za pomoci přerušení. Vybírání procesů k běhu je prováděno periodicky. Procesy jsou ve FreeRTOS systému řazeny do lineárních seznamů dle priorit. FreeRTOS obsahuje tolik lineárních seznamů, kolik je v systému priorit. Obsahem těchto listů jsou procesy, které jsou připraveny k běhu. Každý záznam listu obsahuje ukazatel na

TCB, ukazatel na předcházející záznam a následující záznam, ukazatel na list, ve kterém je záznam uložen a další.

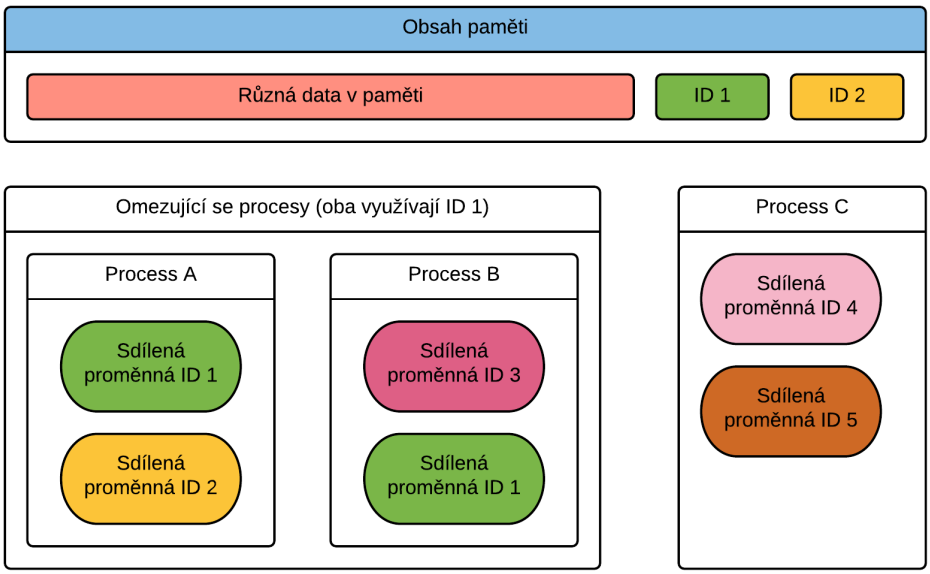
```
 struct xLIST_ITEM
{
  listFIRST_LIST_ITEM_INTEGRITY_CHECK_VALUE
  // Hodnota zaznamu, dle teto hodnoty je cely list usporadan
  // (sestupne, vzestupne)
  configLIST_VOLATILE TickType_t xItemValue;
  // Ukazatel na nasledujici listovy zaznam
  struct xLIST_ITEM * configLIST_VOLATILE pxNext;
  // Ukazatel na prechazejici listovy zaznam
  struct xLIST_ITEM * configLIST_VOLATILE pxPrevious;
  // Ukazatel na object (TCB) ktery obsahuje tento listovy zaznam
  void * pvOwner;
  // Ukazatel na list ve kterem je tento listovy záznam ulozen
  void * configLIST_VOLATILE pvContainer;
  listSECOND_LIST_ITEM_INTEGRITY_CHECK_VALUE
};
```

Plánovač určí za následující proces ten, který se nachází na začátku nejvíce prioritního neprázdného listu. Tím je zaručen výběr úlohy s nejvyšší prioritou a s nejbližším deadlinem.

Synchronizace

V jednojádrovém systému je synchronizace zaručena prostým zakázáním přerušení při práci se sdíleným zdrojem. Proces, který se nachází v kritické sekci (část programu, ve kterém je zakázáno přerušení a pracuje se se sdíleným zdrojem), nemůže být jiným procesem ohrožen ani při použití preemptivního plánovače. Pokud by tato implementace synchronizace zůstala beze změny, byl by výlučný přístup ke sdílenému zdroji zaručen pouze na jednom jádru a ostatní jádra by mohla libovolně přistupovat k tomuto sdílenému prostředku. Možnou úpravou je možné rozšířit kritickou sekci přes všechna jádra, ale tento přístup zcela omezí podstatu použití vícejádrového prostředí. Pokud by se nacházelo jakékoliv jádro procesoru v kritické sekci, ostatní jádra by musela na toto jádro čekat, tzn. po dobu kritické sekce bude vykonávat instrukce pouze jedno jádro.

Možným řešením je označení všech sdílených zdrojů v systému, které jsou využívány jednotlivými procesy. Identifikátor těchto sdílených zdrojů musí být unikátní. Pomocí tohoto unikátního identifikátoru je umožněno procesu, který se nachází v kritické sekci, blokovat pouze ty procesy, které pracují se stejnou sdílenou proměnou.



Obrázek 3.14: Ukázka synchronizace na MC systému

Úloha ve FreeRTOS

Úloha je na platformě podporující dynamickou alokaci vytvořena zavoláním funkce xTaskCreate. Tato funkce nejprve naalokuje TCB a zásobník úlohy. Posloupnost alokací TCB a zásobníku je pěvně určená, je-li zásobník úlohy naalokován jako první, nemůže se již rozšiřovat. Pokud by se zásobník rozšiřoval, začal by přepisovat samotnout datovou strukturu TCB. Pořadí se určuje na základě nastavení FreeRTOS systému. Samotná alokace je prováděna funkcí pvPortMalloc, která používá malloc() a free() funkce poskytnuté kompilátorem. Ve vícejádrovém prostředí je nutné použít tuto konfiguraci, protože tyto funkce jsou bezpečné pro jádro (heap_3). Do TCB úlohy se uloží ukazatel na zásobník úlohy a zaznamená se zda došlo k dynamické či statické alokaci. Poté se zavolá inicializační funkce úlohy prvInitializeNewTask(). Ta naplní a nainicializuje zásobník a TCB blok. Ukazatel pxCreatedTask nyní ukazuje na právě vytvořenou úlohu z důvodu snadnější modifikace parametrů poslední vytvořené úlohy. Posléze se tato úloha přidá do seznamu připravených úloh pomocí funkce prvAddNewTaskToReadyList.

Pokud není vytvořena uživatelem žádná úloha, je vytvořena speciální idle úloha, která běží na procesoru, dokud není plánovačem přerušena. Plánovač je spuštěn pomocí funkce vTaskStartScheduler, ten vytvoří idle úlohu a vybere úlohu ke spuštění. Ukazatel na aktuálně vykonávanou úlohu je uložen v ukazateli pxCurrentTCB.

Task Control Block (TCB)

Struktura TCB je alokována pro každou úlohu a ukládá její stavové informace zahrnující i ukazatele na kontext úlohy. TCB je používán zejména plánovačem, který díky TCB dokáže jednoznačně identifikovat úlohu v systému. Datová struktura TCB obsahuje jméno úlohy, ukazatel na zásobník úlohy, prioritu úlohy a čas, který úloha strávila ve stavu Running a mnoho dalšího viz. důležité části struktury TCB níže.

```
typedef struct tskTaskControlBlock
{
  // Ukazatel na vrchol zasobniku
  volatile StackType_t  *pxTopOfStack;
  // Stav listoveho zaznamu procesu (Running, Suspended)
  ListItem_t      xStateListItem;
  // Tato promenna se vyuziva k referenci ulohy z listu udalosti
  ListItem_t      xEventListItem;
  // Priorita ulohy
  UBaseType_t     uxPriority;
  // Ukazatel na pocatek zasobniku
  StackType_t     *pxStack;
  // Nazev ulohy
  char        pcTaskName[ configMAX_TASK_NAME_LEN ];
  // Ukazatel na konec zasobnik u archutektu, kde se zasobnik rozsiruje
  #if ( portSTACK_GROWTH > 0 )
    StackType_t   *pxEndOfStack;
  #endif
  // Tato promenna udrzuje informaci o poctu vnoreni kriticke sekce
  #if ( portCRITICAL_NESTING_IN_TCB == 1 )
    UBaseType_t   uxCriticalNesting;
  #endif
  // Pri poziti mutexu je nutne udrzovat informaci o
  // posledni pridelene priorite uloze
  // Tato informace je pouzita mechanisme invertovani priorit
  #if ( configUSE_MUTEXES == 1 )
    UBaseType_t   uxBasePriority;
    UBaseType_t   uxMutexesHeld;
  #endif
  // Promenna uklada dobu, po kteoru uloha byla ve stavu Running
  #if( configGENERATE_RUN_TIME_STATS == 1 )
    uint32_t    ulRunTimeCounter;
  #endif
} tskTCB;
```

V jednojádrovém systému je v ukazateli pxCurrentTCB uložen TCB aktuálně vykonávané úlohy. Díky tomuto ukazateli plánovač ví, kterou úlohu bude při přepnutí kontextu nahrazovat. Po nahrazení aktualizuje obsah ukazatele pxCurrentTCB. Při použití vícejádrového prostředí by plánovač s každou změnou kontextu aktualizoval tento ukazatel a tím jej invalidoval pro ostatní jádra. Tento ukazatel je definován jako PRIVILEGED_DATA TCB_t * volatile pxCurrentTCB = NULL;, jeho obsahem je ukazatel pouze na jediný blok TCB. Změna spočívá v převedení ukazatele pxCurrentTCB na pole ukazatelů typu TCB_t o velikosti počtu jader procesoru - 1 (indexace od nuly). Touto změnou vznikne vlastní ukazatel pxcurrentTCB pro každé jádro a každé jádro přistupuje pouze do svého ukazatele na aktuálně běžící proces. Indexem do tohoto pole je ID procesoru.

Přiřazení úlohy jádru

V části diplomové práce 3.2.1 je zmíněno vytvoření speciální úlohy idle, která je vykonávána, pokud není žádná jiná úloha v systému nebo žádná úloha není připravena k běhu. V jednojádrovém systému všechny úlohy běží na stejném jádru, není tedy nutné nijak řešit přiřazení úloh jednotlivým jádrům. To se ale mění ve vícejádrovém systému, kde každé jádro musí mít svou idle úlohu. V tomto případě je nutné upravit definici struktury TCB tak, aby bylo možné rozlišit na jakém jádru má úloha běžet. Do struktury se přidá proměnná o typu integer, která bude určovat, na jakém procesoru daná úloha má běžet a plánovač musí tuto informací brát v úvahu při výběru další úlohy k běhu.

Do struktury TCB je nutné přidat proměnnou UBaseType_t CpuCoreAssigned, jejímž obsahem bude buď identifikátor jádra, nebo konstanta NO_CPU_ASSIGNED, která musí být větší, než počet jader v systému. S touto změnou silně souvisí změna v implementaci plánovače.

Změny v plánovači

V jednojádrovém systému plánovač plánuje úlohy pouze v rámci jednoho jádra. Při použití ve vícejádrovém prostředí by úlohy běžely pouze na prvním jádře a na ostatních by žádná úloha neběžela (ani idle úloha). Je nutné upravit plánovač tak, aby při svém chování reflektoval nastavení úlohy, umožňující si definovat na jakém jádru může být úloha spuštěna. Tento údaj je uložen v datové struktuře TCB úlohy. Pro každé jádro musí být také vytvořena idle úloha (pokud v systému není žádná) s využitím výše uvedených změn.

Při plánování musí plánovač plánovat úlohy určené pouze pro jeho jádro. Nejprve si zjistí, na jakém jádru běží a poté plánuje úlohy, které jsou určeny pro jádro daného plánovače nebo úlohy, které můžou běžet na každém jádře. Plánovač musí využívat při plánování kritické sekce.

Kapitola 4

Zhodnocení SMP a AMP systémů

SMP systémy nejsou cíleny pouze na oblasti vyžadující paralelní výkon, ale i na běžné uživatele. Právě konvenční počítače a procesory (firmy AMD, Intel) pracují v SMP módu. Tyto systémy jsou využity i v mobilních telefonech, které dnes obsahují až 8 jader. Nicméně SMP řešení je pro Real-Time oblast relativně novým pojmem. Real-Time operační systémy většinou toto řešení neimplementují a jsou cíleny pouze na jednojádrové systémy (FreeRTOS, uC/OS-II atd.). Je možné, že některé společnosti si vytvořily vlastní transformaci těchto operačních systémů tak, aby jim operační systém vyhovoval. Tyto transformace jsou ale spíše tajemstvím společností. Proto je častější volbou u Real-Time operačních sytému právě AMP konfigurace, která je snadnější na realizaci.

AMP systémy jsou využívány v mnoho formách, například spuštění dvou různých operačních systémů na každém jádře, nebo spuštění pouze jednoho operačního systému a jedné úlohy běžící přímo na jádře (tj. bare metal úloha). Jádra se dorozumívají většinou pomocí sdílené paměti. Z hediska Real-Time operačních systému je toto řešení nejschůdnější, jelikož je poměrně jednoduché na realizaci. Nejtěžší úlohou v této konfiguraci je zejména komunikace mezi dvěma jádry.

Následující měření jsou provedena na vývojové desce ZedBoard. Během testů SMP konfigurace je využit operační systém Linux. Při testování AMP konfigurace je použita kombinace dvou operačních systémů Linux a FreeRTOS. Při testování je využito UART rozhraní pro sériovou komunikaci a ethernet rozhraní pro komunikace přes ssh protokol. Systémy jsou spuštěny za pomocí U-Boot bootloaderu z SD karty. Použitá SD karta Kingston SDHC 16 GB má přenosovou rychlost při čtení až 90 MB/s a při zápisu až 45 MB/s. Karta je třídy 10, což značí, že minimální průběžná rychlost zápisu je 10 MB/s. V případě SMP konfigurace má systém Linux k dispozici 512 MB, při AMP konfiguraci je paměť rozdělena na polovinu.

4.1 Shrnutí a interpretace výsledků SMP řešení

Největší výhodou, kterou SMP konfigurace přináší, je možnost akcelerace algoritmů pomocí paralelismu. V těchto testech je prezentován problém komprimace souborů s náhodnými daty o velikosti 537 MB. Soubory budou komprimovány pomocí kompresní metody bzip2. V průběhu komprese dat byl měřen proud a průměrná spotřeba ZedBoardu. Spotřeba je měřena pomocí pinů J21. Napětí na pinu J21 je úbytkem na odporu o velikosti 10 miliohmů. Pomocí Ohmova zákona lze vypočítat z těchto dvou hodnot proud, který ZedBoard odebírá. Po vynásobení získaného proudu se vstupním napětím získáme spotřebu desky ZedBoard. V klidovém stavu je průměrná spotřeba desky ZedBoard přibližně 3.872 watů.

Teplota SoC Zynq-7000 je měřena také, v klidovém stavu je teplota čipu přibližně 46.5 stupňů. Následující grafy zobrazují z důvodu přehlednosti pouze 400 sekundová úsek celkové doby komprese dat.

4.1.1 Dostupná jádra v SMP konfiguraci

Pro ověření, zda jsou použita v této konfiguraci opravdu dvě jádra, jsem přečetl obsah souboru /dev/cpuinfo. Část jeho obsahuje je zobrazena níže.

```
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)

processor       : 1
model name      : ARMv7 Processor rev 0 (v7l)

Hardware        : Xilinx Zynq Platform
```

Komprimovací metoda bzip2

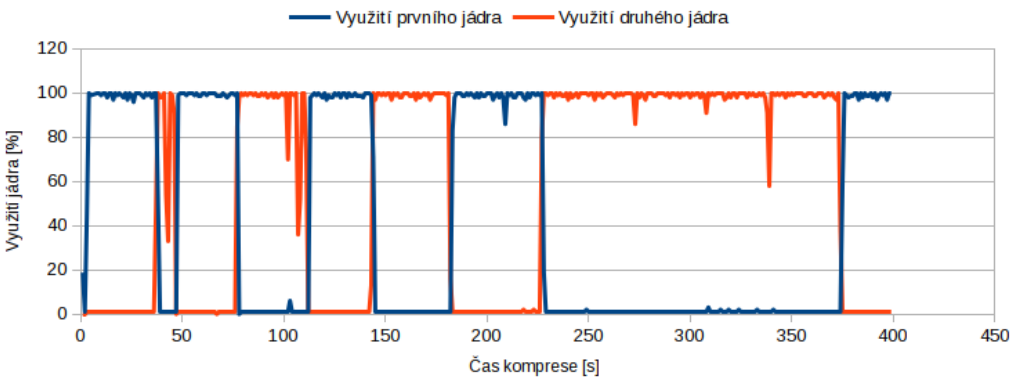
Mezi hlavní znaky komprimovací metody bzip2 patří vysoká míra komprese a v porovnání s ostatními komprimovacími metody rozumná rychlost komprese. Využívá bezztrátovou kompresi dat, což znamená, že při kompresi nejsou ztracena žádná data. Bezztrátová komprese je dosažena použitím Burrows-Wheeler transformace [28].

V původní implementaci metoda bzip2 není uzpůsobena pro paralelní systémy, proto vznikla paralelní verze této metody pbzip2.

Komprimace pomocí jednoho jádra

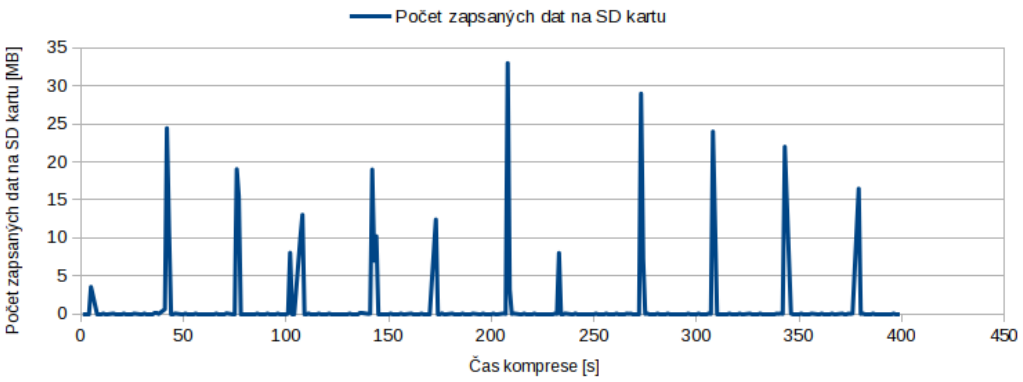
Průměrná teplota SoC Zynq-7000 byla během komprimace 49.6 stupňů a průměrná spotřeba byla 4.598 wattů. Teplota se tedy téměř nezměnila, zatímco spotřeba ZedBoardu se zvýšila o přibližně 700 miliwatů.

Z grafů níže lze vypozorovat plánování jednojádrové úlohy na více jádrech. Plánovač dle plánovacího mechanismu plánuje úlohu tak, aby v jednom okamžiku byla úloha prováděna pouze jedním jádrem. Při změně vykonávacího jádra dochází ke změne kontextu. Celková doba komprimace byla 16 minut a 44 sekund a za tu dobu se v systému vyskytlo 352639 přerušení.



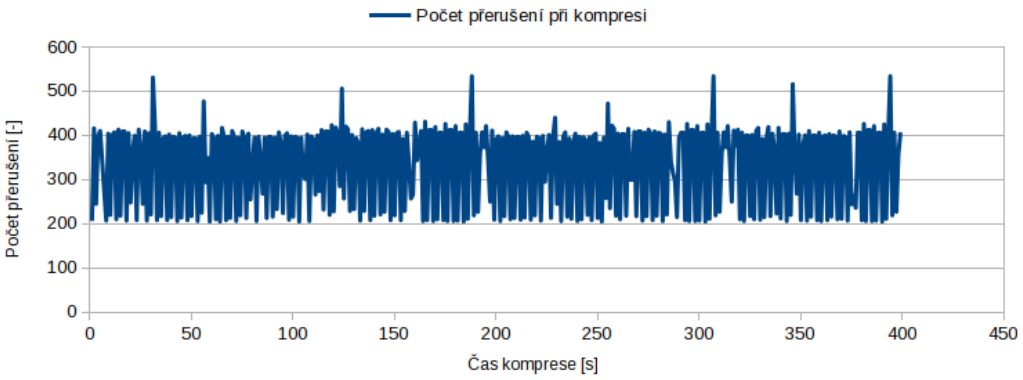
Obrázek 4.1: Využití jader SoC Zynq-7000

Při kompresi byla rychlost zápisu dat na SD kartu průměrně 0.5348 MB/s.



Obrázek 4.2: Množství dat zapsaných na SD kartu

Z důvodu častého využívání přerušovacího systému plánovačem je počet přerušení oproti paralelní verzi vyšší. Při přepnutí kontextu úlohy dochází právě k zaslání přerušení jádru, který úlohy vykonává.

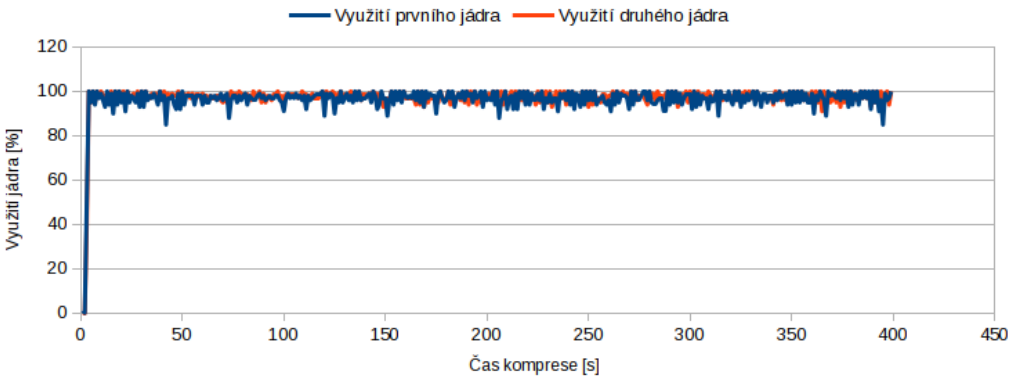


Obrázek 4.3: Počet přerušení v systému

Komprimace pomocí dvou jader

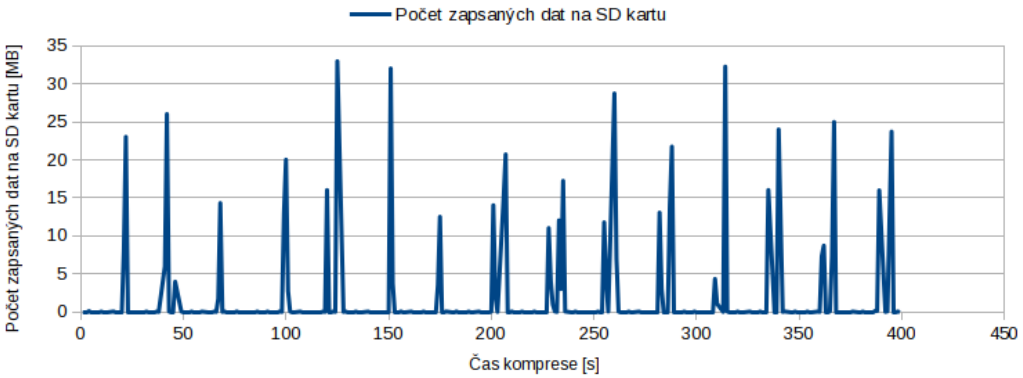
Průměrná teplota SoC Zynq-7000 se zvýšila během komprimace pomocí obou jader na 51 stupňů, což je pouze mírný nárůst oproti neparalelní verzi algoritmu. Průměrná spotřeba byla 4.476 wattů. Spotřeba je oproti neparalelnímu řešení nižší z důvodu výrazně nižší režie při plánování úloh. Nedochází zde tak často ke snižování a zvyšování frekvence procesorů. Oproti neparalelní verzi došlo během vykonávání komprese k výrazně menšímu počtu přepnutí kontextu.

Celková doba komprimace byla 9 minut a 47 sekund, což je téměř poloviční čas oproti neparalelní verzi. Z grafu lze poznat, že po celou dobu komprese byla obě jádra využita na 100%. Celkový počet přerušení v systému byl po dobu komprimace roven 142802, tato hodnota je oproti neparalelní verzi o 60% výrazně menší z důvodu o polovinu rychlejší doby komprimace. Při srovnání počtu přerušení za dobu 9 minut a 47 sekund vyšlo najevo, že se při paralelní komprimaci v systému vyskytlo o přibližně 30000 přerušení méně.



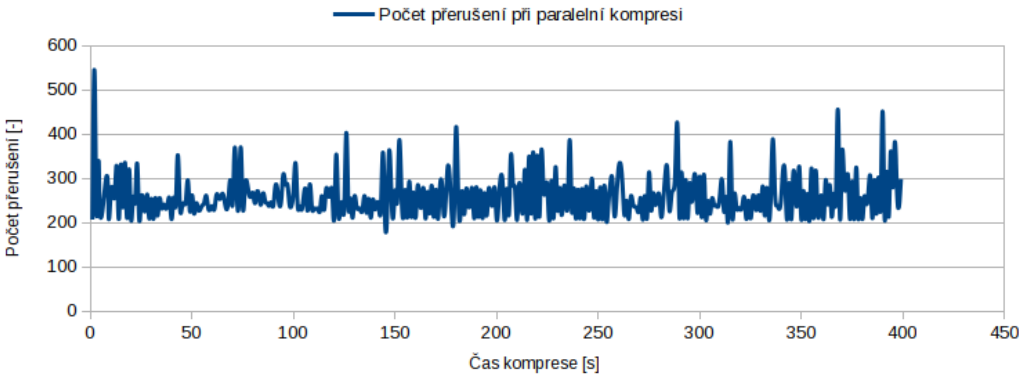
Obrázek 4.4: Využití jader SoC Zynq-7000

Při paralalení kompresi byla průmerná rychlost ukládání dat 0.9418 MB/s. Z testů vychází, procesor Cortex-A9 je prvek, který významně zpomaluje celou kompresi. Přenosová rychlost SD média není ani v jednom případě plně využita.



Obrázek 4.5: Množství dat zapsaných na SD kartu

Plánovač v případě paralelní komprese nemusí tak často přepínat kontext úloh. Z tohoto důvodu je počet přerušení nižší.



Obrázek 4.6: Množství přerušení v systému při paralelní kompresi

4.1.2 Spotřeba systému při použití pouze jednoho jádra procesoru

Tento test byl proveden v prostředí, ve kterém byly povoleny pouze ty periferie, které byly pro běh operačního systému nutné. Spotřeba při vypnutí druhého jádra se téměř nezměnila, byl zaznamenán pokles přibližně o 100 miliwattů. Pokles spotřeby může být významný zejména u aplikací, od kterých je vyžadována energetická úspora. Teplota procesoru se zvýšila o dva stupně.

4.2 Shrnutí a interpretace výsledků AMP řešení

Pro realizaci byla zvolena kombinace operačních systémů FreeRTOS a Linux. Pomocí operačního systému Linux na prvním jádře je možné nainicializovat a nahrát FreeRTOS operační systém na druhé jádro. Komunikace probíhá pomocí ovladače remoteproc, popsáného v 3.1.5. kapitole, který umožňuje komunikaci operačního systému Linux a operačního systému FreeRTOS na druhém jádře. Pro zasílání zpráv je použito rozhraní rpmsg, který využívá soubor zařízení /dev/rpmsg0. Zprávy jsou přes rpmsg toto rozhraní umístovány do VRING bufferu.

4.2.1 Komunikace mezi operačními systémy a měření latence žádostí o přerušení

Pro toto měření byla využita upravená aplikace, původně vytvořená společností Xilinx, demonstrující latence žádosti o přerušení. Žádosti o přerušení jsou měřeny pomocí TTC čítače, který je nastaven tak, aby při přetečení dále čítal. Čítač je po přetečení zastaven obslužnou rutinou přerušení. Hodnota tohoto čítače značí počet tiků hodin od vytvoření požadavku na přerušení. Na straně FreeRTOS jsou ke snímání vzorků použity synchronizační prvky typu semafor.

Součástí tohoto testu je také spuštění periodické úlohy ve FreeRTOS systému. Úloha je probouzena každou sekundu. Při probuzení vypíše zprávu „Periodicka uloha probuzena“. Pokud by se úloha nestihla probudit v požadovaném intervalu, bylo by toto chování zaznamenáno do VRING bufferu.

Dále tento test ukazuje průběh komunikace mezi prvním a druhým jádrem. Komunikace je ustálena zasláním paketu COMM_PACKET a potvrzením tohoto paketu ze strany FreeRTOS.

Průběh komunikace mezi jádry

Následující ukázka je kombinací zpráv z STDOUT výstupu operačního systému Linux a z VRING bufferu operačního systému freeRTOS. Z ukázky lze vyčíst, že minimální zpoždění mezi žádostí o přerušení a obsluhy přerušení bylo 341 nanosekund, nejdelší pak 809 nanosekund. Z důvodu častého výskytu žádostí o přerušení se zpožděním 341 nanosekund je průměrné zpoždení právě 341 nanosekund.

Celkově bylo sesbíráno 22220 žádostí o přerušení.

```
[Druhe Jadro] - Latence - Start vzorkovani IRQ latenci
[Druhe Jadro] - Periodicka uloha spustena
[Druhe Jadro] - Periodicka uloha probuzena.
[Prvni Jadro] - Otevreni RPMSG0 tunelu
[Prvni Jadro] - Otevreni tunelu uspesne
```

```
[Prvni Jadro] - Testovani Komunikace s druhym jadrem
[Druhe Jadro] - Prisel packet COMM_PACKET
[Prvni Jadro] - Ziskan ACK packet od FreeRTOS
[Prvni Jadro] - Komunikace s druhym jadrem navazana
[Druhe Jadro] - Prisel paket START_OVER_PACKET paket
[Prvni Jadro] - Ziskan ACK packet od FreeRTOS
[Prvni Jadro] - Provadim vypocet
[Druhe Jadro] - Latence - Navzorkovano 1000 vzorku
[Druhe Jadro] - Periodicka uloha probuzena.
.............Opakující se zprávy..................
[Druhe Jadro] - Prisel paket STOP_PAKET paket
[Prvni Jadro] - Ziskan ACK packet od FreeRTOS
[Druhe Jadro] - Prisel paket GET_RESULTS_PAKET
[Prvni Jadro] - Ziskan ACK packet od FreeRTOS
[Prvni Jadro] - Tisknu vysledky
[Prvni Jadro] - Zpozdeni 341 ns se vyskytlo 21906 krat
[Prvni Jadro] - Zpozdeni 350 ns se vyskytlo 130 krat
[Prvni Jadro] - Zpozdeni 359 ns se vyskytlo 154 krat
[Prvni Jadro] - Zpozdeni 368 ns se vyskytlo 27 krat
[Prvni Jadro] - Zpozdeni 386 ns se vyskytlo 2 krat
[Prvni Jadro] - Zpozdeni 809 ns se vyskytlo 1 krat
[Prvni Jadro] - Konec Tisku vysledku
[Druhe Jadro] - Periodicka uloha probuzena.
```

Kapitola 5

Závěr

Tato diplomová práce byla zaměřena na problematiku přechodu jednojádrového na vícejádrový operační systém. Dále jsem v této diplomové práci ukázal, jak lze spustit AMP a SMP konfiguraci na zvoleném realizačním prostředku, tj. ZedBoard.

V první kapitole jsem vymezil základní pojmy týkající se problematiky řešení této práce, vytvořil jsem přehled dostupných vícejádrových operačních systémů a přehled vícejádrových platforem. Platformy byly zvoleny dle jejich vlastností. Dále jsou v této kapitole obecně popsány základní funkce jádra a operačního systému. V poslední podkapitole je nastíněna obecná problematika přechodu z jednojádrového na vícejádrové prostředí. Druhá kapitola obsahuje popis zvolených realizačních prostředků této diplomové práce, tj. vývojový kit ZedBoard a operační systém freeRTOS. V kapitole Realizace jsem se zabýval již konkrétními úpravami nutnými ke zprovoznění daných typů multiprocessingu. Při použití AMP konfigurace bylo nutné opravit hardwarový návrh procesorového systému Zynq-7000 tak, aby všechny spuštěné operační systémy měly přístup k požadovaným periferiím procesorového systému. Operační systémy pro AMP přímo nevyžadují úpravy, ale je nutné naimplementovat komunikaci mezi jádry. K tomu byla využita již existující implementace ovladače remoteproc. U SMP řešení jsou projednány jednotlivé změny oproti jednojádrové verzi tohoto operačního systému. Změny byly nutné téměř ve všech oblastech operačního systému. Z důvodu náročnosti změn je realizována pouze jejich část. Změny byly realizovány s minimálním dopadem na API operačního systému FreeRTOS. Ačkoli jsou tyto změny částečně naimplementovány, hlavním problémem je počáteční komunikace jader před spuštěním operačního systému FreeRTOS. Jádra by měla správně nainicializovat přerušovací podsystém a oznámit svůj stav pomocí sdílené paměti. Tento krok jsem ale nedokázal realizovat. Z důvodu popsaných výše jsou testy v kapitole Zhodnocení SMP a AMP systémů spíše porovnávacího charakteru.

Diplomovou práci hodnotím jako přínosnou ale velmi obtížnou. Mezi přínosy této práce řadím zejména práci s procesorem Zynq-7000, s kterým jsem nikdy dříve nepracoval, a hlubší porozuměním funkcím operačního systému. Realizace transformace FreeRTOS byla pro mě velmi obtížná, ale i tak jsem se pokusil o maximální zachování API FreeRTOS a zprovoznění části této transformace alespoň na jednom jádře.

V budoucnu bych se chtěl i nadále věnovat tomuto tématu. V práci bych chtěl pokračovat zejména kvůli vytvoření fungující vícejádrové verzi FreeRTOS operačního systému a jejím umístěním na internet. Pokud by se mi tato realizace povedla, byl by to první volně dostupný port FreeRTOS systému pro SMP konfiguraci na procesoru Zynq-7000.

Literatura

[1] *Antergos Linux*. [Online; navštíveno 25.11.2016].
URL https://antergos.com/about/

[2] *Desktop Operating System Market Share*. [Online; navštíveno 25.11.2016].
URL
https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0

[3] *Mint Linux*. [Online; navštíveno 25.11.2016].
URL https://www.linuxmint.com/about.php

[4] *Historie operačních systémů Windows, Unix, Mac OS a Linux*. Březen 2006, [Online; navštíveno 19.4.2017].
URL http://www.mujmac.cz/rubriky/polemiky/historie-operacnich-systemu-windows-unix-mac-os-a-linux-55713cz

[5] *Embedded Debian Project*. 2014, [Online; navštíveno 19.11.2016].
URL http://www.emdebian.org

[6] *Wind River Linux 8*. 2015, [Online; navštíveno 19.11.2016].
URL http://www.windriver.com/products/product-overviews/WR-Linux-7-Product-Overview.pdf

[7] ArchLinuxArm.org: *Arch Linux Arm*. 2009 - 2016, [Online; navštíveno 19.11.2016].
URL https://archlinuxarm.org

[8] ARM: *ARM Cortex-A53 MPCore Processor*. 2013, [Online; navštíveno 18.4.2017].
URL http://infocenter.arm.com/help/topic/com.arm.doc.ddi0500d/DDI0500D_cortex_a53_r0p2_trm.pdf

[9] ARM: *Zynq-7000 All Programmable SoC Software Developers Guide*. 2015, [Online; navštíveno 17.2.2017].
URL https://www.xilinx.com/support/documentation/user_guides/ug821-zynq-7000-swdev.pdf

[10] ARM: *Zynq-7000 All Programmable SoC Technical Reference Manual*. 2016, [Online; navštíveno 12.12.2016].
URL https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[11] AvNet: *ZedBoard Getting Started Guide*. 2017, [Online; navštíveno 11.11.2016].
URL http://zedboard.org/sites/default/files/documentations/GS-AES-Z7EV-7Z020-G-V7-1.pdf

[12] Barry, P.; Crowley, P.: *Modern Embedded Computing: Designing Connected, Pervasive, Media-Rich Systems*. Elsevier Science, 2012, ISBN 9780123944078.
URL https://books.google.cz/books?id=shCAeVFX6NAC

[13] Barry, R.: *RTOS - Free professionally developes and robust real time os for small embedded systems*. 2016, [Online; navštíveno 16.2.2017].
URL http://www.freertos.org

[14] Bovet, D.; Cesati, M.: *Understanding the Linux Kernel*. O'Reilly Series, O'Reilly, 2002, ISBN 9780596002138.
URL https://books.google.cz/books?id=9yIEji1UheIC

[15] Buildroot: *The Buildroot user manual*. [Online; navštíveno 19.11.2016].
URL https://buildroot.org/downloads/manual/manual.html

[16] Chopra, R.: *Operating System (A Practical App)*. S. Chand Limited, 2009, ISBN 9788121931649.
URL https://books.google.cz/books?id=aaPIP3rP1A0C

[17] Cottet, F.; Delacroix, J.; Kaiser, C.; aj.: *Multiprocessor Scheduling*. John Wiley & Sons, Ltd, 2003, ISBN 9780470856345.
URL http://onlinelibrary.wiley.com/book/10.1002/0470856343

[18] Dionne, D. J.; Durrant, M.: *Embedded Linux/Microcontroller Project*. 2016, [Online; navštíveno 10.11.2016].
URL http://www.uclinux.org

[19] Fox, R.: *Linux with Operating System Concepts*. CRC Press, 2014, ISBN 9781482235906.
URL https://books.google.cz/books?id=zKnNBQAAQBAJ

[20] Furber, S.: *ARM System-on-chip Architecture*. Addison-Wesley, 2000, ISBN 9780201675191.
URL https://books.google.cz/books?id=J_Fu_YTVD9gC

[21] Gargenta, M.: *Learning Android*. O'Reilly Media, 2011, ISBN 9781449307240.
URL https://books.google.cz/books?id=oMYQz4_BW48C

[22] Hajduch, M. O.: *57.1 Operační systémy*. [Online; navštíveno 25.11.2016].
URL http://dum.hajduch.net/VY_32_INOVACE_1ICT9roc_57_B

[23] HardKernel: *Odroid XU4*. Září 2013, [Online; navštíveno 19.1.2017].
URL http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825

[24] Hord, R.: *Parallel Supercomputing in MIMD Architectures*. Taylor & Francis, 1993, ISBN 9780849344176.
URL https://books.google.cz/books?id=FilCw3qBO4gC

[25] Jiménez, M.; Palomera, R.; Couvertier, I.: *Introduction to Embedded Systems: Using Microcontrollers and the MSP430*. SpringerLink : Bücher, Springer New York, 2013, ISBN 9781461431435.
URL https://books.google.cz/books?id=kcHBAAAAQBAJ

[26] LINFO: *Kernel Definition*. Květen 2004, [Online; navštíveno 1.11.2016].
URL http://www.linfo.org/kernel.html

[27] Linfo: *Context Switch Definition*. 2006, [Online; navštíveno 14.11.2016].
URL http://www.linfo.org/context_switch.html

[28] Linfo: *The bzip2 Command*. 2006, [Online; navštíveno 3.5.2017].
URL http://www.linfo.org/bzip2.html

[29] Love, R.: *Linux Kernel Development*. Developer's Library, Pearson Education, 2010, ISBN 9780768696790.
URL https://books.google.cz/books?id=3MWRMYRwulIC

[30] Mouser Electronics, I.: *Zedboard Zynq-7000 Development Board*. Leden 2017, [Online; navštíveno 1.3.2017].
URL http://eu.mouser.com/new/digilent/digilent-zedboard-zynq-7000/

[31] Novell: *Processes in an Uninterruptible Sleep (D) State*. Listopad 2012, [Online; navštíveno 24.11.2016].
URL https://www.novell.com/support/kb/doc.php?id=7002725

[32] NXP: *SABRE Development System*. 2017, [Online; navštíveno 8.5.2017].
URL http://www.nxp.com/products/software-and-tools/hardware-development-tools/sabre-development-system:SABRE_HOME

[33] NXP: *SABRE Platform for Smart Devices Based on the i.MX 6 Series*. 2017, [Online; navštíveno 8.5.2017].
URL http://www.nxp.com/products/archived-tools/sabre-platform-for-smart-devices-based-on-the-i.mx-6-series:RDIMX6SABREPLAT

[34] Olorunosebi, J.: *Memory Management in Multi-Core Systems*. Únor 2016, [Online; navštíveno 24.11.2016].
URL http://resources.intenseschool.com/memory-management-in-multi-core-systems/

[35] Petříček, L.: *Jak vzniká procesor aneb procesorová kuchařka*. Září 2009, [Online; navštíveno 18.1.2017].
URL http://www.svethardware.cz/jak-vznika-procesor-aneb-procesorova-kucharka/14725

[36] Polanka, J.: *Linuxové distribuce*. [Online; navštíveno 25.11.2016].
URL http://home.zcu.cz/~jpolanka/SemprZPS/web/distribuce.htm

[37] RaspberryPi: *RaspBerry Pi Model 3*. Září 2016, [Online; navštíveno 19.1.2017].
URL https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[38] Reddy, C.: *Operating Systems Made Easy*. Laxmi Publications Pvt Limited, 2009, ISBN 9788131807439.
URL https://books.google.cz/books?id=-TdCZplbLUEC

[39] Sharma, E.; Varshney, E.; Sharma, S.: *Design and Implementation of Operating System*. Laxmi Publications Pvt Limited, 2010, ISBN 9789380386416.
URL https://books.google.cz/books?id=3iE3UWrE4L0C

[40] Skopal, J.: *Měření zatížení systému v Embedded Linuxu*. 2014, [Online; navštíveno 24.11.2016].
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=114954

[41] Steiner, G.: *Heterogeneous Multiprocessing: What Is It and Why Do You Need It?* Březen 2016, [Online; navštíveno 15.3.2017].
URL https://forums.xilinx.com/t5/Xcell-Daily-Blog/Heterogeneous-Multiprocessing-What-Is-It-and-Why-Do-You-Need-It/ba-p/696049

[42] Thakur, A.: *Embedded Linux : Understanding the embedded linux*. 2012, [Online; navštíveno 25.11.2016].
URL http://www.engineersgarage.com/articles/what-is-embedded-linux

[43] Thouvenin, G.: *Understanding the structure task_struct*. 2004, [Online; navštíveno 19.11.2016].
URL http://www.spinics.net/lists/newbies/msg11186.html

[44] Verma, S.: *krishna's Operating System*. Krishna Prakashan, ISBN 9788187224914.
URL https://books.google.cz/books?id=DuXvCF0EfDIC

[45] Walls, C.: *Multicore basics: AMP and SMP*. Březen 2014, [Online; navštíveno 6.2.2017].
URL http://www.embedded.com/design/mcus-processors-and-socs/4429496/Multicore-basics

[46] Xilinx: *PetaLinux Tools Overview*. [Online; navštíveno 24.2.2017].
URL http://www.wiki.xilinx.com/PetaLinux

[47] Xilinx: *PetaLinux Tools User Guide*. 2014, [Online; navštíveno 3.3.2017].
URL https://www.xilinx.com/support/documentation/sw_manuals/petalinux2014_2/ug980-petalinux-board-bringup.pdf

[48] Xilinx: *OpenAMP Framework for Zynq Devices*. 2016, [Online; navštíveno 3.3.2017].
URL https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_2/ug1186-zynq-openamp-gsg.pdf

[49] Xilinx: *Zynq-7000 All Programmable SoC*. Leden 2017, [Online; navštíveno 2.3.2017].
URL https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html

Seznam obrázků

Seznam Zkratek

AES - Standard pro šifrování
AMP - Symetrycký MultiProcessing
BSD - Softwarová distrubuce typu Berkeley
DM - Real-Time plánovač Deadline Monotonic
DMA - Přímý přístup do paměti
EDF - Real-Time plánovač Earliest Deadline First
eMMC - Vestavěná multimediální karta
FPGA - Programovatelné hradlové pole
GNU GPL - GNU General Public License
GPIO - Víceúčelové vstupně výstupní piny
GP registr - Víceúčelový registr
LCD - Displej z tekutých krystalů
LLF - Real-Time plánovač Least Laxity First
MCU - Mikrokontrolér
MIMD - Multiple Instruction Multiple Data
MIPS - Architektura procesorů
MMU - Jednotka správy paměti
PID - Identifikátor procesu v systému Linux
RAM - Random Access Memory
ROM - Read Only Memory
RM - Real-Time plánovač Rate Monotonic
RTOS - Real-Time operační systému
SHA - Kryptografická hašovací funkce
SIMD - Single Instruction Multiple Data
SMP - Asymetrický MultiProcessing
SoC - Socket on Chip
UART - Rozhraní pro sériovou komunikaci

Obsah CD

Přiložené CD obsahuje následující soubory a adresáře:

- Složka Technické zprávy obsahující zdrojový kód tohoto dokumentu a jeho překladů do PDF
- Složka mcFreeRTOS, která obsahuje pozměněný kód operačního systému FreeRTOS