Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

ID študenta: 48035

Ing. Martin Vojtko

Písomná práca k dizertačnej skúške

Téma dizertačnej práce:	**Formálny opis vnorených operačných systémov**
Školiteľ:	**doc. Ing. Tibor Krajčovič, PhD.**
Miesto plnenia individuálneho študijného plánu:	**Fakulta informatiky a informačných technológií**
Forma štúdia:	**denná**
Začiatok štúdia:	**1. 9. 2012**
Študijný program:	**aplikovaná informatika**
Študijný odbor:	**9.2.9 aplikovaná informatika**

Dátum: 22. 11. 2013

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Ing. Martin Vojtko

FORMÁLNY OPIS VNORENÝCH OPERAČNÝCH SYSTÉMOV

Písomná príprava k dizertačnej skúške

Študijný program:	Aplikovaná informatika
Študijný odbor:	9.2.9 Aplikovaná informatika
Miesto vypracovania:	Ústav počítačových systémov a sietí
Vedúci práce:	doc. Ing. Tibor Krajčovič, PhD.

November 2013

ANOTÁCIA

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Aplikovaná informatika

Autor: Ing. Martin Vojtko
Dizertačná práca: FORMÁLNY OPIS VNORENÝCH OPERAČNÝCH SYSTÉMOV
Vedúci práce: doc. Ing. Tibor Krajčovič, PhD.
November 2013

Táto práca sa zaoberá formálnym opisom operačných systémov určených pre vnorené systémy. V práci je možné nájsť charakteristiku vnorených operačných systémov a ich triedenie. Jadrom práce je zhrnutie problematiky formálneho opisu v kontexte súbežného návrhu hardvéru a softvéru. Súčasťou zhrnutia je aj výber najpoužívanejších modelov pre opis hardvéru i softvéru. Dôležitou časťou práce je aj zhrnutie metód optimalizácie spotreby energie v hardvéri a softvéri.

ANNOTATION

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: Applied informatics

Author: Ing. Martin Vojtko
Doctoral Thesis: FORMAL DESCRIPTION OF AN EMBEDDED OPERATING SYSTEMS
Supervisor: doc. Ing. Tibor Krajčovič PhD.
2013, November

Topic of this work is a formal description of operating systems intended for embedded systems. In the book can be found characteristic of embedded operating systems. Main part of the work focuses on a formal description in the context of hardware/software co-design. Most used models, which are used in design of hardware and software, are summarised in this work. Power and energy efficiency of the system are taken into account in formal description.

Obsah

Zoznam obrázkov

1 Úvod

Súčasný trend komponentového programovania, využíva predom navrhnuté knižnice, aplikačné používateľské rozhrania a volania operačného systému k tvorbe používateľských aplikácií. Vývojár nedokáže ovplyvniť funkčnosť daných komponentov. Ak komponenty dostatočne nepokrývajú mechanizmy hardvéru, tak vývojár nedokáže tieto mechanizmy využiť bez cieleného a bezpečného obídenia komponentov, ktoré občas ani nie je možné. Vlastnosti aplikačných používateľských rozhraní sú tvorené požiadavkami trhu, nie logikou.

Abstrakcia pomáha návrhárom skrývať zložitosť problému. Pri návrhu používateľskej aplikácie môžeme identifikovať niekoľko úrovní abstrakcie od abstraktnej úrovne hardvéru, ktorá modeluje obvod ako systém medzi-registrových prenosov, po abstraktnú úroveň operačného systému, ktorá modeluje vzájomne komunikujúce úlohy spolu so zariadeniami systému. Môžeme hovoriť o reťazi abstrakcie, ktorá má svoje obrovské výhody v uľahčení práce pri návrhu na tých najvyšších úrovniach. Ale ako sme spomínali takáto reťaz má aj svoje slabiny. Napríklad medzi návrhom jadra operačného systému a hardvérom, na ktorý má byť tento systém nasadený, sú minimálne dve vrstvy abstrakcie. Tieto vrstvy zakrývajú špecifiká daného hardvéru, čo znemožňuje efektívne využitie hardvéru. Našou predstavou je tieto špecifiká hardvéru stransparentniť na vyšších úrovniach abstrakcie.

Tu sa dostávame k problematike návrhu vnorených operačných systémov, ktorá by mala byť zacielená na efektívne využívanie možností hardvéru, čo smeruje k návrhu vnorených operačných systémov pomocou formálneho opisu. Formálny opis operačného systému by mal byť podporený formálnym opisom hardvéru, ktorý pokryje a stransparentní všetky možnosti hardvéru. Návrh formálneho opisu vnorených operačných systémov a formálneho opisu hardvéru sú hlavnými cieľmi tejto práce.

Efektívne využívanie návrhu je v priamej spojitosti so spotrebou energie. Vnorené systémy zdvojnásobujú svoju zložitosť každých $18 - 24$ mesiacov [19]. Na druhej strane energetické zdroje zväčšujú svoju kapacitu o 5% ročne [47]. Tento rozvojový nepomer donútil zmeniť pohľad na návrh vnorených systémov. Kedysi prevažujúci maximálny výkon systémov doplnila energetická efektívnosť.

V súčastnosti existuje na úrovni hardvéru množstvo mechanizmov, ktoré razantne znižujú spotrebu vnoreného systému. Problém sa ale presunul do úrovne softvéru, ktorý zaostáva za rozvojom hardvéru. Vnorený systém sa nemôže stať energeticky efektívny pokým mechanizmy znižujúce spotrebu nie sú využité softvérom. Spomínaný formálny opis bude tieto mechanizmy využívať, k návrhu energeticky efektívnejších operačných systémov.

1.1 Organizácia práce

Druhá kapitola sa venuje vnoreným operačným systémom. Stručne sú tu zhrnuté oblasti, do ktorých problematika vnorených operačných systémov zasahuje. Kapitola sa tiež venuje metrikám na základe, ktorých je možné vnorené operačné systémy porovnávať.

Tretia kapitola sa venuje existujúcemu návrhu Modulárneho operačného systému, ktorý bude sprevádzať prácu.

Štvrtá kapitola sa venuje formálnym metódam opisu hardvéru a špecifikácie softvéru. Okrem formálnych metód sa tu nachádza zhrnutie princípov Hardvér/Softvér co-dizajn[1] a modelovania.

Piata kapitola sa venuje energeticky efektívnemu návrhu vnorených systémov. V prvej časti kapitoly je rozobraná problematika výkonovej a energetickej efektívnosti. Druhá časť sa venuje modelu spotreby energie na tranzistore. V tretej a štvrtej časti sú zhrnuté najpoužívanejšie techniky znižovania spotreby systému.

V šiestej kapitole sú stanovené tézy dizertačnej práce a v siedmej, poslednej kapitole čitateľ nájde zhodnotenie práce na dizertačnom projekte.

1.2 Pojmy a skratky

- **API** - Application programming interface, je knižnica volaní a funkcií, ktoré určujú spôsob práce so zariadením alebo softvérom. Programátor teda nemusí programovať túto funkcionalitu.

- **CMOS** - Complementary Metal Oxide Semiconductor, je technológia tvorby logických hradiel založená na komplementárnom zapojení nMOS a pMOS tranzistorov. Takéto zapojenie výrazne znižuje statickú spotrebu obvodu a citlivosť na okolitý šum.

- **CVM** - Communicating virtual machines, je úroveň abstrakcie návrhu, v ktorej je systém opísaný pomocou vzájomne komunikujúcich úloh. Každá úloha predstavuje virtuálny stroj. Okrem úloh, v systéme vystupuje aj jadro operačného systému, ktoré plánuje prístup virtuálnych strojov na reálny hardvér a riadi ich vzájomnú komunikáciu.

- **FIFO** - First in first out, je organizácia radu, kedy prvý zákazník je obslúžený ako prvý.

- **FPGA** - Field-Programmable Gate Array, je programovateľný obvod, ktorý je nastaviteľný pomocou poľa konfigurácií.

[1] Spoločný návrh softvéru a hardvéru, pozri pojmi a skratky

- **Hammingova vzdialenosť** (Hamming distance) W - je vzdialenosť vypočítaná ako počet navzájom rozdielnych bitov dvoch porovnávaných slov.
- **HW/SW co-dizajn** - Spoločný návrh softvéru a hardvéru.
- **Kernel** - jadro je centrálna časť operačného systému, ktorá ma za úlohu najmä riadiť procesy, prístup k zariadeniam a spravovať pamäť.
- **PASS** - Periodic Admissible Sequential Schedule je periodický dosiahnuteľný sekvenčný plán spúšťania aktérov dátového toku, pri ktorom nedochádza k nestabilite a uviaznutiu dátového toku.
- **Podprahové napätie** (Threshold voltage) V_t - je hranica napätia, ktorej prekročením sa spôsobí otvorenie respektíve uzavretie tranzistora.
- **RPU** - Reconfigurable Processing Unit, je obvod umiestnený v systéme s možnosťou rekonfigurácie, zmeny funkčnosti.
- **RTL** - Register-transfer level, je úroveň abstrakcie návrhu, v ktorej je systém opísaný pomocou medzi-registrových prenosov dát. Každý prenos dát môže vykonať zmeny nad dátami.
- **Shunt Odpor** - je rezistor s vysokou presnosťou hodnoty odporu. Hodnota býva zväčša určená na 1Ω. Tento rezistor sa využíva na meranie prúdu zapojením rezistora do série s meraným obvodom. Zmena prúdu spôsobí zmenu napätia na vstupných bodoch rezistora. Podľa vzťahu $U = RI$ je následne možné prúd vyjadriť.

2 Vnorené operačné systémy

Každý operačný systém má za úlohu zapúzdrovať používaný hardvér a poskytovať vhodné rozhranie pre aplikácie a služby používateľov. Celou myšlienkou operačných systémov je odbremeniť návrhára aplikácie od hardvérovo špecifických problémov [34][35]. Konvenčné operačné systémy pozostávajú z niekoľkých úrovní abstrakcie (jadro, API[1], Ovládače). Návrhár na vyššej úrovni abstrakcie používa jednotné rozhranie pre návrh svojej aplikácie pričom vo väčšine prípadov nemusí poznať hardvér.

Vnorený operačný systém poskytuje viac-menej rovnakú abstrakciu pre návrhára, ktorá však nemá viac úrovní. Vnorený operačný systém je zväčša len jadro, ktoré zabezpečuje plánovanie úloh, alokáciu pamäte a riadenie periférií [15]. To ako sa má jednotlivá periféria používať je rozhodnutím návrhára. Zásadný rozdiel je teda v tom, že návrhár musí poznať minimálne správanie hardvéru a navrhnúť ostatné prvky operačného systému svojpomocne.

Ďalším rozdielom oproti konvenčným operačným systémom je platforma, ktorá je zapúzdrovaná. Vnorené systémy sa vyznačujú tým, že majú zväčša menšie pamäťové prostriedky a menší výpočtový výkon. Na druhú stranu zväčša disponujú omnoho väčšou množinou akčných a senzorických prvkov pripojených na širšom spektre periférií. Doba odozvy je častým nárokom pri spracovávaní vstupných a generovaní výstupných signálov. Často nebýva prítomný trvalý zdroj energie čím sa aj spotreba systému stáva kľúčová. Vnorené systémy sú veľmi rôznorodé a dynamicky sa rozvíjajúce. Preto na vnorené operačné systémy sú kladené najmä nároky ako sú malé pamäťové nároky, malá réžia, práca v reálnom čase, znížená spotreba energie, vysoká miera flexibility a prispôsobivosti.

Vnorené operačné systémy sú veľmi rôznorodé a preto je možné identifikovať špecifické odchýlky akými sú napríklad operačné systémy:

- reálneho času,
- s podporou rekonfigurácie,
- pre aplikácie s nízkou spotrebou energie,
- pre viacjadrové vnorené systémy,
- pre mobilné zariadenia.

[1] Application programming interface, pozri pojmi a skratky

2.1 Operačné systémy reálneho času

Systém reálneho času je systém, ktorý musí vykonávať svoju prácu správne a zároveň v presne vymedzenej časovej odozve. Podľa dopadu nedodržania doby odozvy rozdeľujeme systémy reálneho času na:

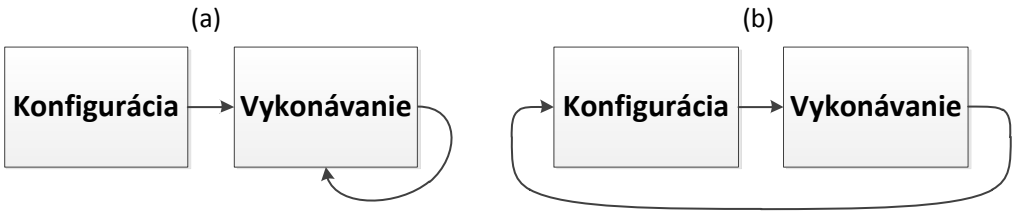
- **Benevolentné systémy reálneho času** (Soft RT) - nedodržanie doby odozvy nemá katastrofálne dôsledky na celý systém. Výsledkom nedodržania doby odozvy je zníženie kvality služby.
- **Striktné systémy reálneho času** (Hard RT) - nedodržanie doby odozvy má katastrofálne následky na celý systém. Výsledkom je narušenie bezpečnosti systému, finančné straty prípadne až ohrozenie životov.

Operačné systémy reálneho času sú operačné systémy s presne vymedzenými dobami odozvy vykonania jednotlivých operácií nad zastrešeným hardvérom. Táto silná podmienka ovplyvňuje celý návrh operačných systémov. Štandardný návrh plánovania úloh, prideľovania pamäte a prideľovania vstupno/výstupných zariadení nie je deterministický, čo je v systémoch reálneho času nutnou podmienkou.

2.2 Operačné systémy s podporou rekonfigurácie

Re-konfigurovateľné systémy sú dôsledkom vyvinutia pro-gramovateľného hardvéru. Programovateľný hardvér ponúka možnosti prípravy obvodov a celých vnorených systémov bez nutnosti tvorby dosky plošných spojov. Programovateľný hardvér bol pôvodne navrhnutý na to, aby urýchlil dobu vývoja nových vnorených systémov. Taktiež umožnil rýchlu tvorbu a ladenie prototypov. Ďalším krokom k rekonfigurácií bolo používanie programovateľných obvodov priamo v prevádzke. Programovateľný obvod poskytuje flexibilitu pri dodatočných úpravách funkčnosti vnoreného systému. Tieto občasné zmeny konfigurácie programovateľných obvodov vyústili k rekonfigurácii. Programovateľný obvod mení svoju konfiguráciu na takú akú systém práve potrebuje [10][43]. Z hľadiska času kedy rekonfigurácia prebieha poznáme [26]:

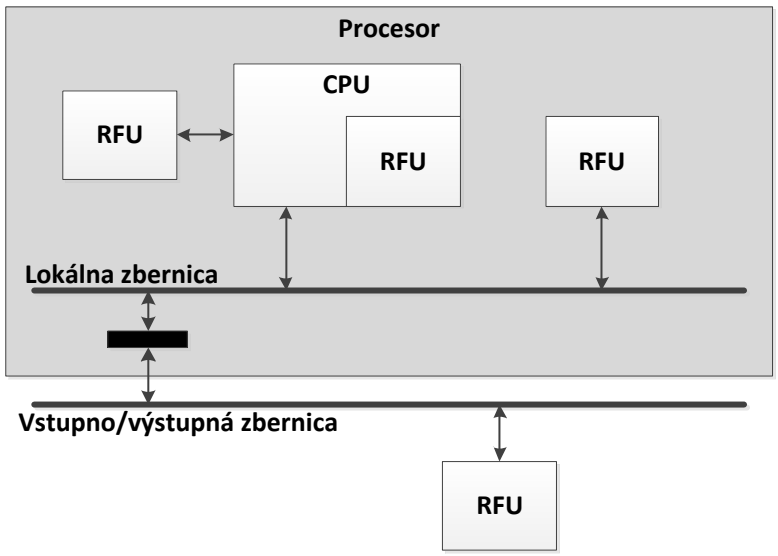
- **Statické** rekonfigurovateľné systémy - rekonfigurácia prebieha v čase, keď systém nie je v prevádzke (obr. 2.1(a)).
- **Dynamické** rekonfigurovateľné systémy - rekonfigurácia prebieha priamo za aktívneho fungovania systému. Rekonfigurácia môže nastať počas priameho používania programovateľného obvodu alebo počas času, kedy je obvod nepoužívaný (obr. 2.1(b)).



Obr. 2.1: Priebeh statickej (a) a dynamickej (b) rekonfigurácie

Z hľadiska umiestnenia programovateľného obvodu vo vnorenom systéme (Obr. 2.2) existuje [43]:

- rekonfigurovateľná **procesná jednotka procesora**[2] (RPU) - umiestnená priamo v procesore.
- rekonfigurovateľný **koprocesor** - pripojený priamo na procesor s exkluzívnym prístupom.
- rekonfigurovateľný **systém na lokálnej zbernici** procesora.
- rekonfigurovateľný **systém na vstupno/výstupnej** zbernici procesora.



Obr. 2.2: Možné umiestnenia rekonfigurovateľnej procesnej jednotky v systéme

Programovateľný obvod je zložený z poľa konfigurovateľných prvkov a pamäte, v ktorej sú uložené jednotlivé konfigurácie obvodu. Na to aby systém zmenil svoju konfiguráciu je

[2]Reconfigurable Processing Unit, pozri pojmi a skratky

nutné zmeniť konfiguráciu v pamäti. Z čoho vyplýva, že zmena konfigurácie trvá značne dlhý čas. Preto nevýhodou rekonfigurácie je dlhá odozva (rádovo v milisekundách).

Napriek tejto nevýhode existujú aplikácie, ktoré využívajú rekonfiguráciu pre svoju prácu. Podobne ako pri plánovaní úloh vykonávaných na procesore je možné plánovať aj rekonfiguráciu [1]. Preto vznikli aj operačné systémy, ktoré podporujú rekonfiguráciu [9][14] [25][29][31].

2.3 Operačné systémy pre systémy s nízkou spotrebou energie

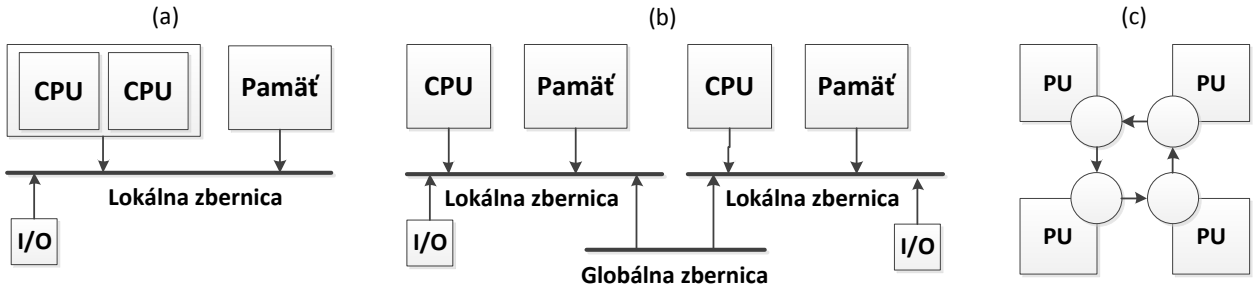
Systémy s nízkou spotrebou sa musia vysporiadať s častými nízkymi úrovňami a výpadkami dodávky energie. Čo ovplyvňuje celý návrh vnorených systémov. Centrálnou požiadavkou na operačný systém je schopnosť zabezpečiť správne spracovanie dát, bezpečné uloženie práce počas odstávky systému a znižovať spotrebu energie na minimum. Počas doby odstávky systému procesor neudržiava svoj stav a preto je nutné jeho stav včas odložiť do pamäte [38]. Počas opätovného spustenia systému je nutné aby operačný systém znovu inicializoval procesor a načítal pôvodný stav systému. Príkladom takéhoto operačného systému je WESPEH OS určený pre bezdrôtové systémy napájané zbieraním energie [38].

2.4 Viacjadrové vnorené systémy

Atraktívnym prístupom návrhu vnorených systémov je rozdelenie viacerých výpočtových problémov a spracovaní do viacerých procesorových jadier. Výhodou takéhoto riešenia je možnosť použitia rôznych procesorových jadier optimalizovaných na konkrétne problémy [11]. Z hľadiska architektúry sa viacjadrové systémy triedia na [37]:

- **Symetrické** - jednotlivé jadrá sú homogénne a pamäť s perifériami si navzájom zdieľajú (obr. 2.3(a)).
- **Asymetrické zbernicové** - jednotlivé jadrá majú vlastnú lokálnu pamäť a periférie pripojené na lokálnu zbernicu. Lokálna pamäť je pre ostatné procesory prístupná cez systémovú globálnu zbernicu (obr. 2.3(b)).
- **Asymetrické sieťové** - jednotlivé jadrá majú vlastnú lokálnu pamäť a periférie pripojené na lokálnu zbernicu. Spoločne tvoria procesnú jednotku PU. Procesné jednotky nemôžu komunikovať prostredníctvom pamäte ale len prostredníctvom výmeny paketov (obr. 2.3(c)).

Každá architektúra má svoje pre aj proti. Symetrická architektúra má vysokú priepustnosť spracovania, ale na druhej strane možnosť vzájomného blokovania si prostriedkov.



Obr. 2.3: Architektúry viacjadrových systémov. (a) symetrická, (b) zbernicová, (c) sieťová

Preto nemusí byť vždy zaručená doba odozvy. Operačný systém nasadený do takejto architektúry rovnomerne zaťažuje jednotlivé jadrá pričom samotný sa vykonáva len na jednom jadre[37].

Asymetrické zbernicové viacjadrové systémy sa používajú v systémoch s reálnym časom. Najčastejšie bývajú heterogénne, kde jednotlivé úlohy sú pevne (staticky) rozložené na jednotlivé jadrá. Jednotlivé jadrá a periférie sú optimalizované na vykonávanie pridelených úloh. Operačný systém musí pracovať na každom jadre samostatne [37].

Asymetrické sieťové viacjadrové systémy majú topológiu siete a komunikácia medzi jednotlivými procesormi je realizovaná pomocou paketového prenosu. Sieťová topológia je vhodná pre systémy, v ktorých sú vykonávané výpočtovo náročné čiastkové úlohy s nižším prenosom dát medzi úlohami [37].

2.5 Operačné systémy pre mobilné aplikácie

Fenoménom posledného desaťročia sa stali inteligentné telefóny, navigácie a iná elektronika bežnej osobnej spotreby. Operačné systémy takejto elektroniky sú značne komplikovanejšie ako štandardné vnorené operačné systémy, ale stále javia znaky vnorených operačných systémov. Najmä preto, že majú len jedného používateľa a špecifické aplikačné rozhrania. Predstaviteľmi takýchto operačných systémov sú Android, Windows CE, Embedded Linux.

2.6 Architektúry vnorených operačných systémov

Z hľadiska usporiadania prvkov v operačnom systéme poznáme štyri základné architektúry vnorených operačných systémov [15]:

- **Monolitické** - navrhnuté ako silne previazaná a závislá štruktúra. Výhodou takejto štruktúry je malá réžia systému, pretože prepojenie zariadenia s procesom je na úrovni volania jednej procedúry. Nevýhodou tejto organizácie je veľmi obtiažne rozširovanie

funkcionality alebo zlepšovanie vlastností OS. Aj malý zásah do systému spôsobí veľké zmeny.

- **Monolitické-modulárne** - sú systémy rozvrhnuté do monolitických modulov. Koncept zjednodušuje ladenie a úpravu systému. Typickým predstaviteľom je Embedded Linux.
- **Vrstvové** - systémy majú štruktúru usporiadanú do vrstiev medzi ktorými je vytvorený systém rozhraní (obr. 2.4(a)). Vrstva poskytuje služby vyššej vrstve a žiada služby od vrstvy nižšej. Výhodou takéhoto konceptu je jednoduchá úprava systému. Nevýhodou je nabalujúca sa réžia na každej vrstve systému, ktorá môže vo vnorených systémoch byť kľúčová. Typickým predstaviteľom je FreeDOS.
- **Mikro-kernel**[3] - navrhnuté s čo najjednoduchším jadrom spravujúcim zväčša len procesy, pamäť a vstupno/výstupné zariadenia (obr. 2.4(b)). Podtriedou mikro-kernel operačných systémov sú nano-kernel systémy, ktoré majú v jadre implementované len riadenie procesov a vstupno/výstupných zariadení.

Obr. 2.4: Architektúra operačných systémov. (a) vrstvová, (b) mikro-kernel

Okrem základných architektúr existujú rôzne kombinácie, z ktorých vrstvovo-modulárna architektúra ktorej predstaviteľom je *Modulárny operačný systém (MOS)* [40][42]. Myšlienky *Modulárneho Operačného Systému* sú stručne objasnené v kapitole 3.

[3] pozri pojmi a skratky

2.7 Metódy porovnávania vnorených operačných systémov

Na určenie rozdielov a predností rôznych vnorených operačných systémov je nutné zaviesť metriky, ktorými sa jednotlivé operačné systémy dajú porovnávať. V závislosti od požadovaných vlastností operačného systému sa používajú nasledujúce metriky [18]:

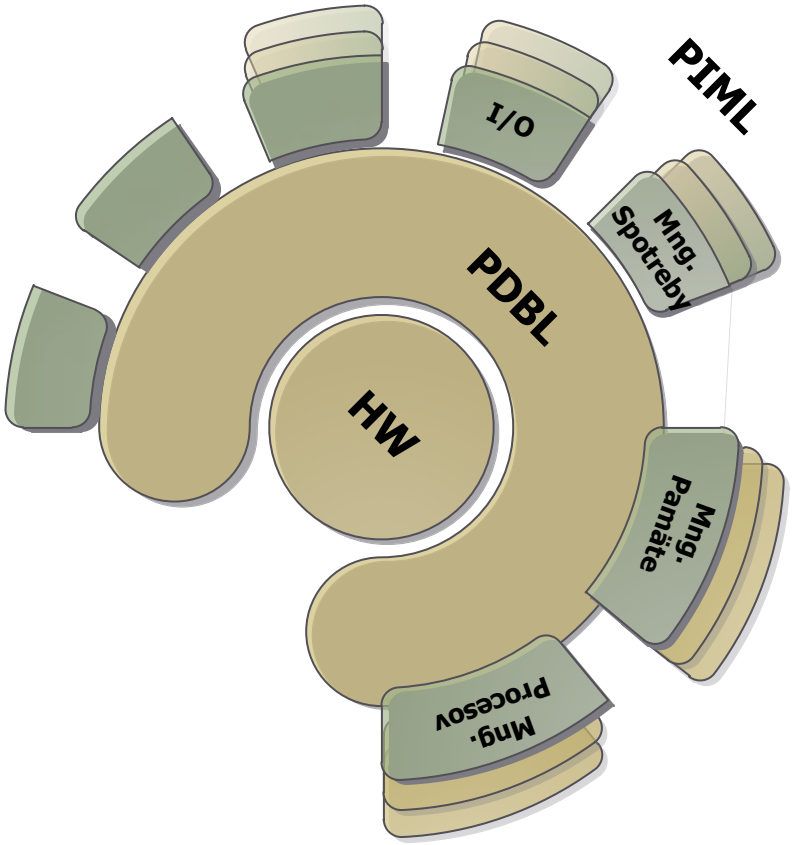
- réžia prepnutia úlohy,
- latencia prerušenia,
- latencia plánovania úloh,
- použitý mechanizmus plánovania úloh,
- použitý mechanizmus správy pamäte,
- čas do rozpoznania uviaznutia,
- mechanizmus prístupu do kritických oblastí.

V štúdiách [20][39][44] je podrobnejšie rozpracované porovnanie rôznych operačných systémov určených pre bežné aplikácie. Použitá metodika sa čiastkovo dá aplikovať aj na vnorené operačné systémy.

3 Modulárny operačný systém

Z hľadiska realizácie sa pre nás za najefektívnejšiu architektúru vnoreného operačného systému javí vrstvovo-modulárna architektúra. Príspevok vrstvovej architektúry je rozčlenenie operačného systému na 2 a viacej hladín abstrakcie. Členenie do vrstiev uľahčuje správu a rozvoj operačného systému. Modulárna architektúra umožňuje rozčlenenie jednotlivých vrstiev do samostatne stojacích problémov a rozšíriť prispôsobivosť operačného systému. Na druhú stranu vrstvovo-modulárna architektúra prináša oneskorenie spracovania úloh a zvýšenie réžie operačného systému. Je preto dôležité stanoviť optimálny počet vrstiev a optimálnu granularitu modulov.

Modulárny Operačný Systém (MOS) je príkladom vrstvovo-modulárnej architektúry (obr. 3.1) [42]. Tento operačný systém je prototypom vnoreného operačného systému, ktorý navrhujeme.



Obr. 3.1: Architektúra Modulárneho operačného systému

Pri návrhu MOS sme sa rozhodli rozdeliť jeho štruktúru do dvoch vrstiev (obr. 3.1). Prvá vrstva (platformovo-závislá PDBL[1]) zapúzdruje hardvér a poskytuje platformovo-nezávislé

[1]Platform Dependent Base Layer

rozhranie vyššej vrstve. Ako taká nenadobúda žiaden stav. Táto vrstva je implementovaná v jazyku symbolických inštrukcií konkrétnej platformy. Niektoré časti môžu byť implementované aj vo vyššom programovacom jazyku C.

Druhá vrstva (modulárna PIML[2]) pozostáva z jednotlivých modulov, z ktorých každý má za úlohu zapúzdrovať vybranú časť hardvéru, prípadne môže slúžiť ako aplikačná podpora pre používateľské aplikácie. Modulárna architektúra umožňuje mapovanie komponentov vnoreného operačného systému na konkrétne komponenty procesora a vnoreného systému. Tým sa dá dosiahnuť efektívne využitie hardvéru a zároveň nastaviť len tie časti systému, ktoré sú potrebné pre riešenie problému.

Operačný systém v našom ponímaní je rámcom napĺňaným modulmi. Každý modul môže byť realizovaný vo viacerých verziách s ohľadom na rôzne parametre, ako sú doba odozvy, spotreba pamäte, výkon, úspora energie.

Počas skúmania problematiky vnorených operačných systémov sme zistili, že aparát pre formálny opis vnoreného operačného systému na systémovej úrovni je nepostačujúci. Preto by bolo prínosom pre oblasť skúmania zaviesť formálny opis vnorených operačných systémov. Formálny opis by mal obsahovať opis operačného systému jeho komponentov a vzťahov medzi nimi. Formálny opis by mal vychádzať z formálneho opisu systému ako celku. Súčasne by mal reflektovať architektúru procesora (komponenty a režimy) použitého v systéme [41],[42].

Formálny opis vnoreného operačného systému zo samotnej podstaty vyžaduje aby vnorený operačný systém mal modulárny charakter. Tento fakt ďalej podporuje myšlienku modulárnosti operačných systémov [41],[42],[40].

[2]Platform Independent Modular Layer

4 Formálny opis

V tejto kapitole sa venujeme analýze metód spojených s návrhom hardvéru a softvéru. Vnorený systém, ako taký, pozostáva z hardvéru a softvéru preto úlohou návrhára je zohľadniť tieto dve zložky a zosúladiť ich do takej miery aby plnili svoju úlohu čo najlepšie. Tomuto zosúladeniu sa hovorí hardvérový a softvérový co-dizajn.

Formálny opis operačného systému a formálny opis hardvéru sú predmetom tejto kapitoly. Formálny opis hardvéru chápeme ako opis správania sa a architektúry už existujúceho alebo vyvíjaného systému, na ktorý sa bude nasadzovať operačný systém. Formálny opis operačného systému slúži na opísanie vlastností architektúry a správania operačného systému. Na úrovni návrhu vnoreného operačného systému budeme formálny opis hardvéru používať na opis hardvéru.

Častým spôsobom ako opísať vlastnosti a funkčnosť hardvéru je katalógový list (datasheet). Návrhár v ňom nájde všetky potrebné informácie pre svoj návrh. Myšlienkou formálneho opisu hardvéru je transformácia informácií z katalógového listu do formálnej roviny, ktorá do určitej miery bude môcť automatizovať implementáciu.

Formálny opis operačného systému bude použitý na špecifikovanie architektúry a správania sa operačného systému na konkrétne opísanom hardvéri. Spojením týchto dvoch stavebných kameňov si sľubujeme efektívny návrh a implementáciu operačných systémov šitých na mieru hardvéru.

4.1 Hardvér/Softvér co-dizajn

HW/SW co-dizajn je rozdeľovanie, návrh a implementácia aplikácií do fixných a flexibilných komponentov [30]. Fixné komponenty tvorí nemenný hardvér, alebo v špecifických prípadoch aj softvér. Flexibilné komponenty sú tvorené najme softvérom, ale môže to byť aj programovateľný hardvér.

Primárnymi prvkami HW/SW co-dizajnu sú modely. Konkrétne to je kombinácia modelov hardvéru a softvéru. Model je reprezentáciou systému, v ktorom sú zvýraznené dôležité časti systému. Pri hardvéri to môže byť model RTL[1], ktorý modeluje hardvér ako systém medzi-registrových prenosov. To ako sú registre a logické hradlá implementované je skryté. Softvér môže byť modelovaný programovacím jazykom C. Pričom systém funkcií a premenných kódu odráža chovanie algoritmu, ale to aký inštrukčný súbor používa procesor je skryté.

Najdôležitejšou úlohou návrhára je rozhodnúť, ktoré časti bodú realizované ako hardvér, a ktoré ako softvér. Pri rozhodovaní môže pomôcť povaha navrhovanej aplikácie a požiadavky

[1]Register transfer level, pozri pojmi a skratky

na ňu. V zásade sa jedná o rozhodovanie medzi fixným a flexibilným návrhom. Návrhárovi môžu pomôcť nasledujúce faktory [30]:

- **Výkon** - Počet spracovaných dát za jednotku času rozhoduje o tom, či sa návrhár nakloní viac k fixnému hardvéru alebo flexibilnému softvéru.
- **Energetická efektívnosť** - Spracovanie dát priamo hardvérom je energeticky efektívnejšie ako procesorom vybaveným softvérom.
- **Výkonová hustota** - Taktovacie frekvencie súčasných procesorov dosiahli fyzikálne hranice. Preto sa vývoj začal uberať k viac-procesorovým architektúram. Vývoj softvéru na takéto architektúry je komplikovanejší.
- **Komplexnosť návrhu** - V prípade veľmi komplikovaných aplikácií je jednoduchšie realizovať väčšiu časť aplikácie pomocou softvéru.
- **Cena návrhu** - Jednoduchý hardvér a komplikovanejší softvér je lacnejší ako v opačnom prípade. Návrh hardvéru a jeho implementácia môže pozostávať z niekoľkých časovo náročných krokov.
- **Zužovanie času návrhu** - Nové generácie aplikácie sú z pravidla komplikovanejšie ako staršie. Nárastom komplexnosti sa vlastne ten istý čas na návrh zužuje. Flexibilné aplikácie vo forme softvéru sú teda efektívnejšie ako fixné hardvérové.
- **Problémy nového dizajnu** - Novšie technológie hardvéru prinášajú vyššiu mieru variability a menšiu zavádzaciu spoľahlivosť. Použitie flexibilného návrhu je rýchlejšie pri odstraňovaní problémov s novým návrhom.

4.1.1 Dualizmus hardvéru a softvéru

Hardvér a softvér sú radikálne odlišné návrhové paradigmy. Ich odlišnosti sumarizuje tabuľka 4.1

Tabuľka 4.1: Porovnanie hardvérového a softvérového návrhu [30]

	Hardvér	Softvér
Paradigma	Dekompozícia v priestore	Dekompozícia v čase
Cena	Plocha, počet hradiel	Čas, počet inštrukcií
Obmedzenie	Čas, perióda	Plocha, inštrukčný set
Flexibilita	Musí byť navrhnutá	Implicitná
Paralelizmus	Implicitný	Musí byť navrhnutý
Model	$\neq$ implementácia	$\sim$ implementácia
Znovupoužitie	Nie časté	Časté

Prirodzenosťou hardvéru je paralelizácia, teda súčasné vykonávanie viacerých operácií v jeden časový okamih. Dôležitým parametrom hardvéru je jeho plocha, od ktorej závisí jeho cena, ale aj spotreba energie. Obvod je obmedzovaný hodinovým signálom. Štandardný obvod nemôže zmeniť svoju štruktúru, čím stavia hardvér do fixnej roviny. Flexibilita hardvéru môže byť dosiahnutá použitím špeciálnych obvodov, ako je napríklad FPGA[2]. Model hardvéru sa nedá považovať za výslednú implementáciu.

Softvér na druhú stranu disponuje flexibilitou. Problém je realizovaný formou algoritmu, ktorý je rozprestrený v čase. Cena softvéru je udávaná v dĺžke spracovania úlohy alebo spotreby pamäte. Softvér je ohraničený množinou inštrukcií alebo operácií, ktoré ponúka daná platforma. Realizácia paralelného spracovania úloh musí byť implementovaná napríklad procesmi a vláknami. Model softvéru sa blíži k implementácií postupnými iteráciami.

4.1.2 Paralelizmus a súbežnosť

Rozdielom medzi súbežným a paralelným systémom je v architektúre. Súbežným je systém vtedy, ak pozostáva z prvkov, ktoré môžu byť vykonávané súbežne. Systém je paralelný vtedy, ak súbežné prvky sú vykonávané v rovnakom čase, teda na samostatne vyčlenenom hardvéry. Hardvér z jeho podstaty je paralelný.

Softvér je štandardne sekvenčný. Na to aby bol softvér súbežný (z angl. concurrent) je nutné implementovať podporu z hardvéru ako je prepnutie úloh. Prepnutie úloh zabezpečí odloženie kontextu úlohy do pamäte systému a načítanie inej úlohy. Na to aby bol softvér paralelný je nutné mať systém s viacerými procesnými jednotkami.

Úlohou návrhára je efektívne využiť paralelizmus a súbežnosť v aplikácií. Ak špecifikácia má čisto sekvenčnú povahu nemá zmysel uvažovať o súbežnom prípadne paralelnom vykonávaní. Návrhárovi môže pri rozhodovaní pomáhať Amdahlovo pravidlo, ktoré udáva maximálne zrýchlenie výpočtu ako obrátenú hodnotu podielu sekvenčného kódu programu.

$$Z = \frac{1}{s} \tag{4.1}$$

Ak celkový podiel sekvenčne vykonávaného kódu v aplikácií je $s = 0.25$, tak celkové teoretické zrýchlenie aplikácie môže byť maximálne štvornásobné. Je vhodné rozložiť aplikáciu do štyroch súbežných blokov. Tieto bloky môžu byť vykonávané súbežne na jednom procesore pomocou prepínania úloh alebo na viacerých procesoroch paralelne.

Ďalší dôležitý parameter, ktorý je nutné zvážiť je utilizáca hardvéru, ktorá udáva mieru využitia výpočtovej kapacity procesora, respektíve iného hardvérového komponentu. Napríklad majme aplikáciu s podielom sekvenčného kódu $s = 0.25$. Predpokladajme, že daná aplikácia sa dá rozdeliť na štyri rovnaké úlohy pričom každá čaká na odozvu periférie 50% času, potom celkový čas T vykonávania aplikácie a utilizácia U systému bude:

[2]Field-Programmable Gate Array, pozri pojmi a skratky

- Pre jeden procesor a sekvenčný kód $T = 4, U = 0.5$ pretože celá aplikácia beží v jednom časovom rade, pričom kým neskončí jeden blok nemôže sa vykonať ďalší. Keďže blok polovicu času čaká na perifériu, tak procesor nie je využitý na 100%.
- Pre jeden procesor a súbežný kód $T = 2, U = 1$ ak zanedbáme réžiu prepnutia úlohy. Procesor prepne úlohu ak čaká na perifériu. Keďže úloha čaká 50% času na perifériu tak čas čakania využije druhá úloha.
- Pre dva procesory a súbežný kód $T = 1$,$U = 1$ Štyri úlohy sa rozdelia na dva procesory.

V uvedených príkladoch predpokladáme, že periféria vie obsluhovať všetky požiadavky od úloh súčasne a úlohy medzi sebou nekomunikujú. Ak by to tak nebolo tak je potrebné zohľadniť aj tieto vplyvy vo výpočte.

4.2 Modelovanie

Model je abstrakciou systému, ktorý slúži na vyjadrenie pre návrhára dôležitých vlastností systému. Vlastnosti systému, ktoré nie sú v konkrétnom momente dôležité, model skrýva. Na opis systému existuje množstvo modelov. Môže to byť [3]:

- **Koncepčný model** - ktorým analyzujeme systém viac v problémovej oblasti ako v oblasti riešenia. Takýmito modelmi sú napríklad Petriho sieť, Stavový automat, diagram toku dát a iné. Tieto modely sa vyznačujú nezávislosťou od času.
- **Fyzicky model** - ktorým vieme manipulovať. Príkladom je obvod FPGA, ktorým modelujeme hardvér.
- **Matematický model** - ktorý je vyjadrený algoritmom.
- Vizuálny model, ktorý zobrazuje ako sa systém správa v reálnom alebo simulovanom svete.
- **Logický model** - ktorý vyjadruje štruktúru systému. Tieto modely analyzujú systém v oblasti riešenia. Tieto modely sa vyznačujú závislosťou od času.

Z hľadiska abstrakcie času môžeme modeli triediť na [30]:

- **Spojité** - modely nízkej abstrakcie, ktoré opisujú systém ako obvod, kde správanie systému prebieha na vzájomnej interakcii diferenciálnych rovníc. Tieto modely sú príliš podrobné pre návrh HW a SW.
- **Diskrétne** - modely, ktorých aktivita je vyjadrená ako sled nepravidelných udalostí. Nepravidelná udalosť je zmena vstupov obvodu. Model zachytáva oneskorenie prenosu zo vstupu na výstup. Pre HW a SW sú tieto modely na nízkej úrovni abstrakcie.

- **Cyklovo-presné** - modely, ktoré sa opierajú o pravidelný hodinový signál hardvéru. Tieto modely zanedbávajú oneskorenia vznikajúce pri prenose signálu. Všetky udalosti v systéme sú naviazané na hodinový signál. Cyklovo-presné modely sú pri návrhu HW a SW používané veľmi často.

- **Inštrukčne-presné** - modely, ktoré systém zobrazujú ako sled inštrukcií. Inštrukcia sa skladá z jedného alebo viac cyklov systému. Inštrukcia skrýva zmeny všetkých signálov v obvode do jednej operácie. Tieto modely sa používajú na modelovanie a verifikovanie komplexných softvérových systémov, akými sú aj operačné systémy.

- **Transakčné** - modely, ktorých aktivita je vyjadrená vzájomnou interakciou komponentov modelu. Komponent môže byť aj softvér aj hardvér, čim sa transakčné modely stávajú veľmi nápomocnými pri súčasnom návrhu hardvéru a softvéru. To či je komponent hardvér alebo softvér ešte nemusí byť pevne vyšpecifikované.

4.2.1 Výpočtový model

Je model, ktorým sa snažíme dokázať, že systém ktorý je modelovaný, je schopný riešiť danú úlohu [3]. Príkladom výpočtového modelu je napríklad procesor. Procesor poskytuje konkrétne operácie, ktoré dokáže vykonať. Zároveň umožňuje uložiť vstupy a výstupy operácií v registroch alebo v hlavnej pamäti procesora. Na základe presne vymedzených operácií teda vieme odhadnúť a predpokladať správanie programu, ktorý bude na danom type procesora spracovaný. Návrhár následne dokáže analyzovať správanie programu a hodnotiť jeho efektívnosť bez ohľadu na vnútornú realizáciu procesora. Napríklad čas vykonania jednotlivých inštrukcií nie je potrebný v počiatočnom hodnotení efektivity programu.

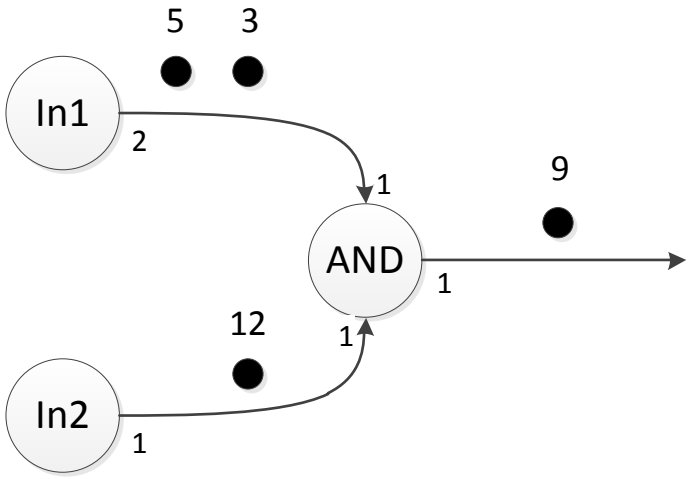
Výberom výpočtového modelu vkladáme do návrhu systému určitú mieru implementačného rozhodovania. Napríklad výberom modelu procesora s jedným výpočtovým jadrom rozhodujeme o sekvenčnej povahe riešenia systému. Teda v jeden čas je vykonávaná práve jedna inštrukcia. Rovnako ak modelujeme algoritmus pomocou programovacieho jazyka C, tak je zrejmé, že daný algoritmus bude sekvenčný pretože jazyk C vychádza z Von Neumannovej architektúry, ktorá má čisto sekvenčnú povahu. Na to aby sme mohli algoritmus navrhnúť súbežne je nutné využiť implementované knižnice, ktoré sa dajú považovať za doplnok jazyka a nadstavbu pôvodného výpočtového modelu.

Možnosti modelu sú mapované jazykom. Jazyk je syntaxou výpočtového modelu zatiaľ čo výpočtový model udáva sémantiku. Napríklad procesor a jeho možnosti sú pokryté jazykom symbolických inštrukcií alebo pomocou jazyka vyššej úrovne ako je napríklad jazyk C. Jazyk C, ale aj jazyk symbolických inštrukcií sú textovej povahy. Okrem textu môžeme použiť aj grafický zápis modelu ako je to napríklad pri stavovom automate.

4.2.2 Model toku dát

Softvérový model (napríklad v jazyku C) nie je vhodným prostriedkom na modelovanie hardvéru rovnako ako hardvérový model (napríklad RTL) nie je vhodným prostriedkom na modelovanie softvéru. Oba modely vnášajú do návrhu systému implementačné rozhodnutia, čo je v HW/SW co-dizajne nežiaduce. Je nutné použiť taký model, ktorý nevnáša implementačné rozhodnutia, ale môže pomôcť rozhodnúť ako sa systém bude ďalej deliť do hardvérových a softvérových blokov. Takýmto modelom môže byť Model toku dát.

Model toku dát (obrázok 4.1) je graf, ktorého vrcholy sú aktéri systému a hrany sú rady. V radoch sa prenášajú značky, ktoré symbolizujú prenášané dáta.



Obr. 4.1: Príklad toku dát logického súčtu

Aktéri predstavujú operácie, ktoré sú vykonané nad vstupnými dátami (značkami). Každý aktér má presne vymedzený začiatok a koniec vykonávania operácie. Jedno vykonanie aktéra sa nazýva spustenie (z angl. fire) a teoreticky trvá nekonečne krátky čas. Aktér je spustený ak má na každom zo vstupov požadovaný počet značiek. Pre prípad z obrázku to znamená, že Aktér AND sa spustí ak má na každom vstupe aspoň jednu značku. Aktér vyprodukuje toľko značiek koľko má vyšpecifikovaných na výstupe.

Rady tvoria jednosmerný tok dát medzi dvoma aktérmi. Transportujú značky z počiatočného aktéra ku koncovému aktérovi. Rad je typu FIFO[3] a má teoreticky nekonečnú dĺžku, teda nemôže dôjsť ku strate dát.

Výhody modelu toku dát sú [30]:

- Nezávislosť na poradí spracovania dát. Sú spracované tie dáta, ktoré sú práve pripravené. Teda tok dát je súbežný a preto sa dá jednoducho transformovať na sekvenčný softvér alebo paralelný hardvér.

[3]First in first out, pozri pojmi a skratky

- Distribuovanosť. Tok dát nepotrebuje centrálne riadenie. Aktéri teda môžu byť implementovaný paralelne ako hardvér alebo sekvenčne ako softvér.
- Jednoduchá analýza modelu. Problémy ako uviaznutie a stabilita môžu byť hodnotené inšpekciou modelu.
- Deterministickosť. Tok dát je deterministický ak všetci aktéri implementujú deterministickú funkciu. Ak aktér implementuje deterministickú funkciu nezáleží na tom ako je implementovaný.

V návrhu systémov sa najčastejšie používa takzvaný synchrónny tok dát. Na rozdiel od obyčajného toku dát má každý aktér synchrónneho toku dát presne zadefinované a nemenné počty konzumovaných a produkovaných značiek. Nemenné počty konzumovaných a produkovaných značiek umožňujú hľadanie takého poradia (plánu) spúšťania aktérov toku dát, pri ktorom nedochádza ku nekonečnému hromadeniu značiek (nestabilite) v žiadnom z radov (obrázok 4.2 vľavo) a zároveň nedochádza k takému výslednému značkovaniu, v ktorom sa nespustí žiaden ďalší aktér (uviaznutiu) (obrázok 4.2 vpravo) [30].

Obr. 4.2: Príklad nestabilného toku dát vľavo a uviaznutého toku dát v pravo.

Po návrhu dátového toku systému prichádza na rad hľadanie takého plánu spúšťania aktérov pre ktorý platí, že je stabilný, prebieha do nekonečna a aktéri sa vykonávajú pravidelne v rovnakom poradí. Takýto plán spúšťania nazývame Periodický dosiahnuteľný sekvenčný plán (PASS) [16].

Postup hľadania Periodického dosiahnuteľného sekvenčného plánu PASS[4] odvodili páni Lee a Messerschmidtt v roku 1987 [16]. Tento postup pozostáva zo štyroch krokov:

1. Transformácia toku dát na topologickú maticu $G(m, n)$. Kde m je počet hrán a n je počet vrcholov. Prvok matice G_{ij} nadobúda kladnú hodnotu ak vrchol j produkuje značky do hrany i a naopak zápornú hodnotu ak vrchol j konzumuje značky z hrany i.

[4]Periodic Admissible Sequential Schedule, pozri pojmi a skratky

2. Ak nemá matica hodnosť o jedna menšiu ako počet aktérov dátový tok nemá Periodický dosiahnuteľný sekvenčný plán.

3. Určenie takého spúšťacieho vektora, ktorý nenarúša stabilitu dátového toku q_{PASS}. Je to vektor, ktorý určuje počet spustení jednotlivých uzlov počas jednej periódy. Taký vektor po násobení s maticou G produkuje nulový vektor. Takýto vektor voláme periodický spúšťací vektor. Keďže hodnosť matice G je o jedna menšia ako počet vrcholov tak existuje nekonečne veľa takýchto vektorov. Existencia periodického vektora q_{PASS} negarantuje PASS.

4. Určenie poradia spúšťania, ktoré produkuje PASS. Cieľom je nájsť také poradie spúšťania aktérov aby sa každý spustil q_{PASS} vektorom určený počet krát.

Problémom dátového toku je jeho nekonečný priebeh Aktéri sú vždy spustený ak majú príslušný počet vstupných značiek. Dátový tok je fixný a bez zastavenia hardvéru a jeho rekonfigurácie nie je možné vykonať zmenu. Dátový tok s obtiažami modeluje výnimky. Vykonávanie vetvení je rovnako veľmi problematické. Preto vzniklo množstvo odnoží synchrónnych dátových tokov, ktoré tieto problémy riešia ale sú mimo pôsobnosti tejto práce.

4.2.3 Konečný stavový automat

Konečný stavový automat je sekvenčný digitálny stroj charakterizovaný množinou stavov, množinou vstupných a výstupných slov, prechodovou funkciou a výstupnou funkciou.

Najjednoduchšia reprezentácia stavového automatu je graf. Jednotlivé vrcholy grafu predstavujú možné stavy, do ktorých sa systém môže dostať. Hrany predstavujú povolené prechody medzi stavmi. Prechodová funkcia vyjadruje povolené prechody medzi stavmi automatu na základe vstupov automatu. Výstupná funkcia predstavuje aktuálny výstup automatu na základe aktuálneho vstupu a stavu systému (Mealyho stroj), respektíve len stavu (Moorov stroj).

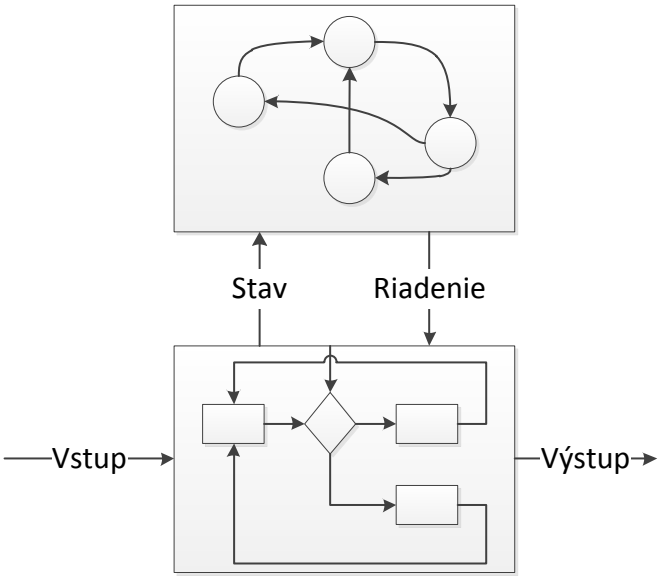
Dĺžka prechodu jedným stavom nie je pre model ako taký dôležitá. Tá sa stáva dôležitou pri implementácií. Napríklad pri implementácií riadiacich jednotiek logických obvodov to býva štandardne jedna perióda hodinového signálu.

4.2.4 Konečný stavový automat s dátovou cestou

Konečný stavový automat s dátovou cestou je snahou oddeliť funkčný výpočet od riadenia výpočtu. Tento model teda logicky aj štruktúrne oddeľuje riadiacu časť systému od operačnej.

Riadenie je veľmi jednoducho opísateľné stavovým automatom, zatiaľ čo operačná časť je jednoduchšie opísateľná výrazom logickej funkcie alebo algoritmom (obrázok 4.3). Na

druhú stranu opísať riadenie logickou funkciou alebo opísať operačnú časť stavovým automatom môže byť obtiažne.



Obr. 4.3: Schéma stavového automatu s dátovou cestou.

4.2.5 Model komunikujúcich virtuálnych strojov

Predpokladajme, že sme vytvorili vnorený systém s príslušným operačným systémom. Na modelovanie vzájomne komunikujúcich úloh spustených na navrhnutom systéme je vhodné použiť model komunikujúcich virtuálnych strojov (Communicating Virtual Machines, CVM) [8].

Myšlienkou takéhoto modelu je realizovať každú úlohu ako samostatný virtuálny stroj (procesor s virtuálnou pamäťou). Každý takýto virtuálny stroj využíva jadro operačného systému, ktoré predstavuje abstraktný stroj C (Abstract C Machine)[5]. Jadro operačného systému teda realizuje výmenu aktuálne bežiaceho virtuálneho stroja na reálnom procesore, zabezpečuje komunikáciu medzi virtuálnymi strojmi, obhospodaruje prerušenia a komunikáciu so zariadeniami.

4.3 Formálne metódy opisu hardvéru

Po finalizácií návrhu systému, ako celku, a určení povahy jednotlivých komponentov (hardvér alebo softvér) prichádza na rad implementácia jednotlivých komponentov.

[5]Abstraktný model stroja zapúzdrujúci model procesora. pozri pojmi a skratky

Hardvérové komponenty môžu byť opísané zo stránky správania alebo zo stránky štruktúry. Správanie je opisom reakcií komponentu na okolie bez toho aby sme vedeli ako odozvy na okolie sú vnútorne realizované. Štruktúra je opisom vnútornej kompozície prvkov, ktoré sú opísané správaním. Pri zložitých systémoch štruktúrny opis môže dosahovať aj niekoľko úrovní.

V tejto práci sa zameriame na stránku opisu už existujúcich komponentov systému, ktorý sa v zásade nelíši od opisu ešte len navrhovaných komponentov.

4.3.1 Opis správania

Podstatou opisu správania systému je zvýraznenie odozvy systému a nie jeho realizácie. Systém opísaný správaním je teda pre návrhára čiernou skrinkou.

Vhodným modelom opisu správania je teda stavový automat. Stavový automat mapuje reakciu vnútorného stavu komponentu na vstup a zároveň ním vieme modelovať reakciu výstupu na aktuálny stav a vstup komponentu.

Ďalším často používaným spôsobom opisu správania systému je časový diagram. Časový diagram vyjadruje závislosť prechodu vstupu na stav a výstup komponentu.

4.3.2 Opis štruktúry

RTL model je predstaviteľom opisu štruktúry hardvérového komponentu. Štruktúra vyjadruje vzájomné prepojenie vnútorných prvkov komponentu. Opis štruktúry hardvéru je vhodný pri návrhu nových komponentov. Nový komponent je vyjadrený kompozíciou už existujúcich komponentov nižšej úrovne. Ak ale realizujeme vzájomnú spoluprácu existujúceho hardvérového komponentu s inými komponentami, tak štruktúra komponentu sa stáva menej dôležitou.

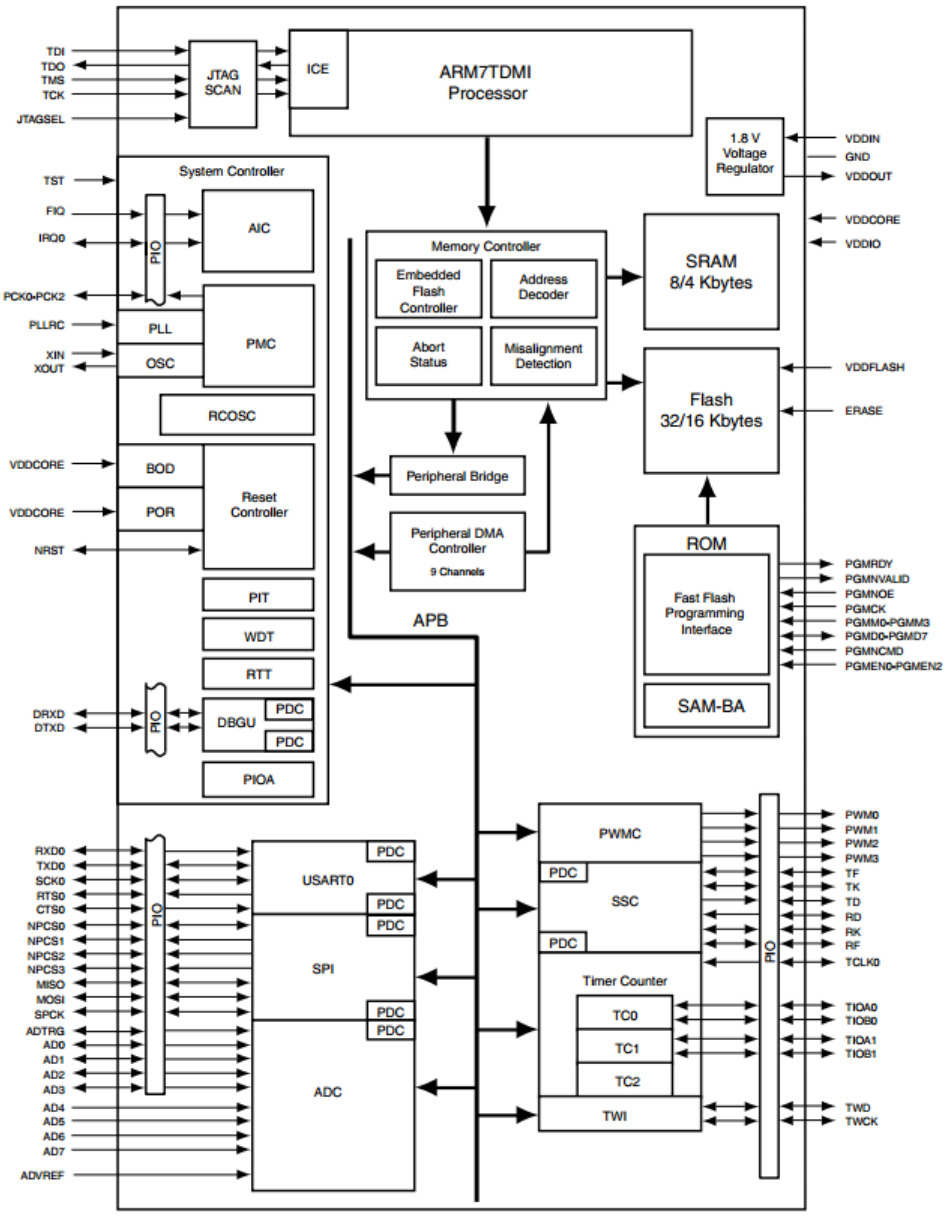
4.3.3 Od dátového listu ku formálnemu opisu

Každý výrobca hardvéru dodáva k svojmu produktu aj dátový list (z angl. datasheet). Úlohou dátového listu je informovať používateľa o možnostiach daného hardvéru. V prvom rade sa jedná o fyzikálne vlastnosti hardvéru, príklady zapojenia a iné. Súčasťou dátového listu je:

- **Schematický opis štruktúry** komponentu uvedený najčastejšie vo forme blokovej schémy.
- **Funkčný opis správania** komponentu, ktorý vyjadruje reagovanie komponentu na vstupy z okolia. Správanie býva opísané časovým diagramom, konečným stavovým automatom, slovným opisom. Dôležitými informáciami pre návrhára sú najme doby odozvy komponentu.

- **Používateľské rozhranie**, ktoré opisuje spôsob komunikácie s komponentom. Pod komunikáciou môžeme rozumieť bežnú výmenu vstupov a výstupov a konfiguráciu, nastavenie režimu komponentu. Opis používateľského rozhrania obsahuje vstupy, výstupy a konfiguráciu komponentu.

Ak sa jedná o komplikovanejší hardvér, akým je napríklad procesor, tak dátový list môže zlučovať opis štruktúry, správania a používateľského rozhrania viacerých prvkov. Na obrázku je príklad štruktúry procesora *at91sam7s* [2]. Daný procesor obsahuje procesné jadro *ARM7tdmi* a iné periférie. Jednotlivé periférie procesora sú podrobnejšie opísané v samostatných kapitolách dátového listu.



Obr. 4.4: Architektúra procesora at91sam7s256 [2].

Častou súčasťou dokumentácie k hardvérovému komponentu bývajú aj podporný firmvér a príklady použitia, ktoré sú nápomocné najmä vo fáze zoznamovania sa s platformou. Najmä firmvér je dôležitý zo strany návrhu pretože jeho súčasťou sú napríklad hlavičkové súbory špecifikujúce adresy registrov platformy prípadne až do úrovne bitov. Firmvér zväčša pomôže používateľovi nakonfigurovať komponent do najbežnejšej formy správania.

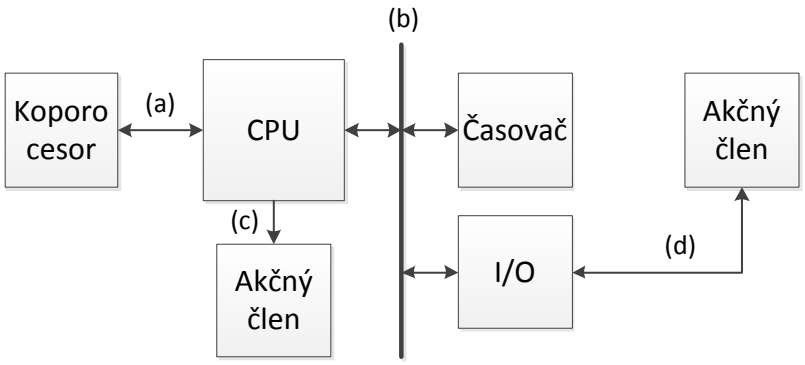
Dátový list je formou opisu hardvéru, ktorá je čitateľnou pre návrhára. Našou myšlienkou je transformovať tento opis do formy čitateľnej pre počítač alebo presnejšie návrhový prostriedok, ktorý bude ďalej pracovať so získanými informáciami. Pre formálny opis je dôležitý najme opis správania hardvérového komponentu a opis používateľského rozhrania.

Používateľské rozhranie

Používateľské rozhranie udáva spôsob komunikácie s hardvérovým komponentom. V softvérovom ponímaní to je časť kódu, ktorá je silno platformovo závislá a teda z hľadiska vrstvového modelu operačného systému sa používateľské rozhranie prenesie do platformovo závislej vrstvy. V spôsobe ako procesor komunikuje s komponentom je možné identifikovať dva možné scenáre:

- **Priama komunikácia** prebiehajúca medzi procesorom a komponentom bez aktívneho sprostredkovateľa. Príkladom takejto komunikácie je komunikácia koprocesor-procesor (obr. 4.5(a)) cez vyhradený komunikačný kanál, komunikácia periféria-procesor (obr. 4.5(b)) cez adresno-dátovú zbernicu procesora alebo komunikácia periféria-procesor (obr. 4.5(c)) formou nastavenia signálov.

- **Nepriama komunikácia** prebiehajúca medzi procesorom a vzdialeným komponentom s aktívnym sprostredkovateľom (obr. 4.5(c)). Príkladom takejto komunikácie je komunikácia procesora s akčnými a senzorovými prvkami pripojenými na periférii procesora. Procesor využíva konkrétny vstupno/výstupný obvod k obsluhe daných zariadení.



Obr. 4.5: Používateľské rozhrania medzi procesorom a perifériou

Z hľadiska operačného systému je pre priamo pripojený komponent potrebné implementovať kód zapúzdrujúci daný komponent. Pre prípad nepriamo pripojeného komponentu sa môžu brať do úvahy dva možné prístupy.

- Pristupovať k danému vzdialenému komponentu ako k pripojenému zariadeniu, ktoré nie je priamou súčasťou systému. Prístup ku komponentu sa rieši pomocou služieb alebo procesov, spracovaných operačným systémom. Daný proces využíva sprostredkujúci komponent zapúzdrený operačným systémom k riadeniu vzdialeného komponentu.
- Pristupovať k danému komponentu ako ku súčasti systému. Prístup ku komponentu je riešený pomocou modulu, ktorý je súčasťou operačného systému. Modul zapúzdrujúci vzdialený komponent je spätý s modulom sprostredkujúceho komponentu a využíva jeho služby.

Z formalizačného hľadiska je nutné transformovať opis rozhrania z dátového listu do opisu zrozumiteľnému pre aplikáciu transformujúcu rozhranie do riadiaceho kódu.

Opis správania

Opis správania udáva najme stavy aké môže hardvérový komponent nadobudnúť, ale čo je ešte dôležitejšie, v akých časových intervaloch sa tieto stavy môžu meniť. Keďže opis správania má často stavovú povahu je predurčený na transformáciu do platformovo nezávislej vrstvy operačného systému.

4.4 Formálne metódy opisu softvéru

Rovnako ako hardvér aj softvér vieme opísať dátovým tokom alebo stavovým automatom. Ak začneme z najvyšších úrovni, tak môžeme hovoriť o návrhu používateľských aplikácií. Návrhár takejto aplikácie pozerá na systém ako na množinu operácií vo forme volaní operačného systému a aplikačných používateľských rozhraní. Ak postupujeme na nižšie úrovne abstrakcie prechádzame na úroveň operačného systému, ktorý komunikuje s jadrom operačného systému. Ak postupujeme ešte nižšie tak sa dostaneme do jadra, ktoré môže mať viacero navzájom komunikujúcich vrstiev abstrakcie. Najnižšia platformovo-závislá vrstva jadra komunikuje s hardvérom.

4.4.1 Programovací jazyk ako prostriedok formálneho opisu

Aj keď existuje mnoho rôznych spôsobov ako opísať softvér (tok dát alebo stavový automat, Petriho sieť, diagram tried) vždy je tento softvér implementovaný pomocou programovacieho jazyka. Softvér má tu výhodu, že model realizovaný programovacím jazykom sa iteratíve blíži k samotnému riešeniu. Rozdiel v modeli a reálnom nasadení sa postupne stiera.

Ak hladíme na celý systém z vertikálneho pohľadu, môžeme identifikovať viacero vrstiev navzájom spätých modelov (obrázok 4.6). Na najnižšej vrstve to je model preklápacích obvodov a pamäte (model RTL). Následne to je vrstva strojového jazyka (Machine Language Layer), ktorá je modelovaná strojom s náhodným prístupom (Random Access Machine) a k nemu pridruženou množinou inštrukcii. Tieto vrstvy sú obalené vrstvou programovacieho jazyka ako je napríklad jazyk C s príslušným abstraktným prekladačom nazývaným aj Abstraktný stroj C (Abstract C Machine) [8].

Abstraktný stroj C ako model umožňuje modelovať softvér. Rovnako dobre slúži ako prostriedok formalizácie. Príkladom využitia programovacieho jazyka ako prostriedku formálneho opisu je jadro operačného systému CVM^* [8][4].

4.4.2 Formálny opis operačného systému

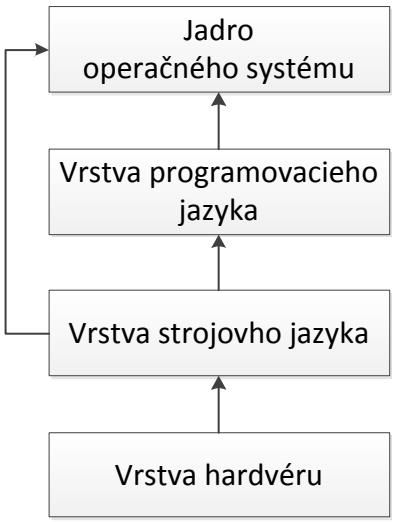
Pre naše potreby uvažujme, že operačný systém je zložený len z jadra. Ostatné komponenty zanedbajme. Stále berieme do úvahy, že realizujeme operačný systém určený pre vnorené aplikácie. V predchádzajúcej kapitole sme uviedli, že programovací jazyk je prostriedkom formalizácie. Takže operačný systém môžeme opísať pomocou programovacieho jazyka C. Využitie programovacieho jazyka má aj jednu výhodu. Tou výhodou je, že programovací jazyk je okrem prostriedku formalizácie a modelovania aj prostriedkom implementačným.

Pri toľkej jednoduchosti ale nesmieme zabúdať na jeden dôležitý fakt. Programovací jazyk vyššej úrovne nemá možnosť pristúpiť k stavovým registrom procesora. Táto možnosť je pred návrhárom skrytá prekladačom. Operačný systém ale potrebuje pri prepnutí úlohy odložiť stav danej úlohy a načítať stav novej úlohy. Tento proces vyžaduje prístup k registrom. Preto je nutné hľadať pomoc o vrstvu abstrakcie nižšie vo vrstve strojového jazyka.

Obrázok 4.6 znázorňuje prechod medzi jednotlivými vrstvami formálneho opisu. Na opis jadra operačného systému potrebujeme formalizačný aparát ponúkaný programovacím jazykom ale zároveň aj strojovým jazykom.

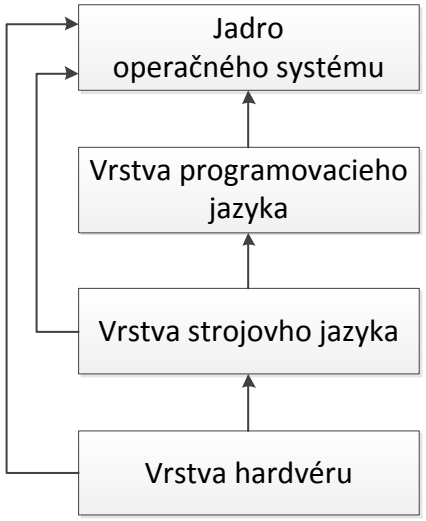
Abstraktná úroveň strojového jazyka modeluje systém ako množinu operácií nad pamäťovým priestorom a stavom procesora. Úroveň programovacieho jazyka tento priestor operácií prenáša do roviny operácií nad pamäťovým priestorom. Takže aj akákoľvek práca s perifériou procesora je realizovaná ako práca nad konkrétnou oblasťou pamäte systému.

Takže ak navrhujeme operačný systém návrhár musí hľadať spojitosť s perifériou pro-



Obr. 4.6: Vertikálne rozčlenenie abstraktných úrovní formálneho opisu

cesora v dátovom liste a preniesť potrebné vlastnosti periférie do kódu, ktorý danú periférium má na starosti. Našou predstavou je túto periférium formálne opísať tak, aby z tohto opisu bolo možné automatizovane generovať riadiaci kód pre operačný systém.



Obr. 4.7: Priviazanie závislosti jadra operačného systému od hardvéru.

5 Spotreba energie

Spotreba energie je kľúčovým parametrom vnorených systémov. Od spotreby systému závisia rozhodnutia návrhára spojené s voľbou zdroja energie, výber architektúry obvodov a mnohé iné. Pri návrhu systému je preto dôležité zohľadniť aj tento parameter. Pri riešení otázky vysokej spotreby už existujúceho vnoreného systému má návrhár dve možnosti. Návrhár môže použiť silnejší zdroj energie. Na druhej strane môže zvážiť možnosti daného hardvéru a použitého softvéru. Je softvér dostatočne optimálny? Využívajú sa možnosti úspory energie hardvéru naplno?

Použitie silnejšieho zdroja energie je jednoduchšie riešenie ale nedá sa aplikovať do nekonečna. Batériové zdroje ročne zvýšia svoje kapacity o 5% [47] zatiaľčo prírastok zložitosti vnorených systémov je dvojnásobný každých 12-18 mesiacov [19]. Z trendu je jasné, že spotreba systémov rastie neudržateľne a preto aplikácia úsporných opatrení je na mieste.

V tejto kapitole sa budeme venovať možnostiam zníženia spotreby energie, ktoré nám poskytuje hardvér a softvér. V danej problematike existuje množstvo riešení, ktoré ale návrhár nedokáže ovplyvniť na existujúcom systéme alebo pri návrhu softvéru. Preto týmto možnostiam sa nebudeme venovať.

5.1 Energeticky a výkonovo efektívny systém

Pojem energeticky efektívny systém sa často krát používa vo význame výkonovo efektívny. Preto je dôležité aby boli tieto pojmy správne vysvetlené pretože sú dôležité z pohľadu návrhu.

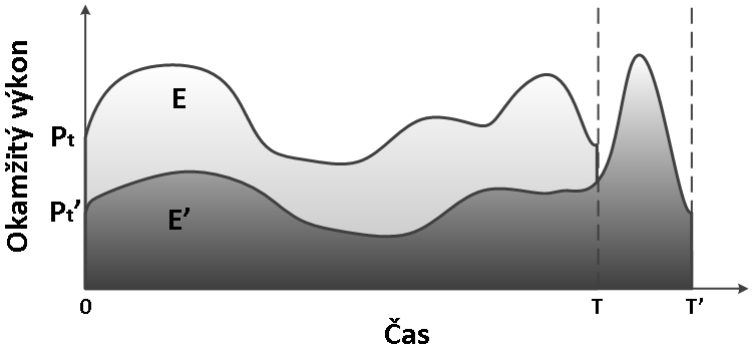
$$P_t = U_t \times I_t \tag{5.1}$$

Podľa rovnice 5.1 môžeme vyjadriť okamžitý výkon P_t ako súčin okamžitého napätia U_t a prúdu I_t.

$$E = \int_0^T P_t \, \mathrm{d}t \tag{5.2}$$

Energia spotrebovaná obvodom je rovná ploche pod funkciou výkonu P na danom intervale $< 0; T >$.

Zo vzťahu 5.1 vyplýva, že systém S je výkonovo efektívnejší ak jeho priemerný výkon P je nižší ako priemerný výkon P' systému S'. Dôležitou vlastnosťou výkonovej efektívnosti je, že nezávisí od času. Na druhej strane zo vzťahu 5.2 vyplýva, že systém S je energeticky efektívnejší ak jeho celková spotrebovaná energia E v čase T je menšia ako celková spotrebovaná energia E' systému S' v čase T'. Dôležitou vlastnosťou energetickej efektívnosti je, že závisí od času.

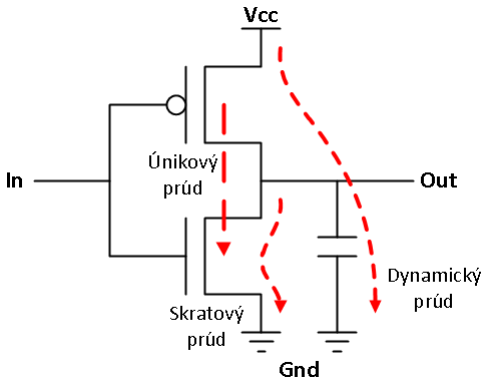


Obr. 5.1: Príklad dvoch systémov S a S'

5.2 Okamžitý výkon systému

Tranzistor je základným stavebným kameňom logického obvodu a je spotrebiteľom energie. V súčastnosti sa pri tvorbe logických obvodov najviac používa technológia CMOS[1].

Výkon na tranzistoroch CMOS podľa obrázku 5.2 sa dá rozdeliť na tri časti. Je to dynamický, skratový a únikový (statický) výkon. Dynamický a skratový (Short-circuit) výkon nadobúda obvod počas zmeny stavu tranzistora. Únikový výkon existuje kým je tranzistor napájaný [24].



Obr. 5.2: Tri zložky spotreby výkonu

Dynamický výkon tranzistora vzniká pri zmene stavu tranzistora. Okamžitý dynamický výkon obvodu P_d sa dá vyjadriť pomocou vzťahu 5.3.

$$P_d = \alpha C V_{cc}^2 f \tag{5.3}$$

Kde α je koeficient udávajúci pomer počtu tranzistorov, ktoré zmenili svoj stav a počtu tranzistorov v obvode, C je parazitická kapacita obvodu, V_{cc} je pracovné napätie obvodu a

[1]Complementary Metal Oxide Semiconductor, pozri pojmy a skratky

f je taktovacia frekvencia obvodu.

Skratový výkon je oproti ostatným zložkám výkonu tranzistora zanedbateľný. Vzniká počas nábehu a dobehu napájacieho napätia na bránach tranzistora. Je priamo úmerný rozdielu medzi časmi nábehu a dobehu medzi vstupnou a výstupnou bránou tranzistora.

Únikový výkon je dôsledkom sumy niekoľkých typov únikov na prepojeniach brán a substrátu tranzistora. Zmenšovaním tranzistora sa tento únikový výkon exponenciálne zväčšuje. Výhodou CMOS technológie bola práve nízka spotreba, ktorá pri súčasne používaných technológiách dosiahla úroveň dynamickej spotreby.

5.3 Optimalizácia spotreby energie na úrovni hardvéru

Problém vysokých energetických nárokov vnorených systémov sa rieši aplikáciou optimalizačných metód na rôznych úrovniach návrhu systému. Od tranzistorovej úrovne po systémovú úroveň. Pre našu prácu sú zaujímavé najmä tie techniky, ktoré vie priamo použiť návrhár na úrovni RTL[2] a úrovni Systémovej.

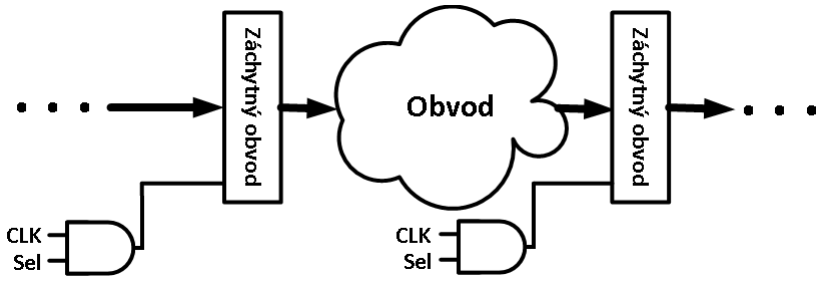
- Znižovanie dynamického výkonu
  - Hradlovanie hodinového signálu.
  - Zníženie frekvencie a napätia obvodu,
- Znižovanie statického výkonu
  - Úsporný režim obvodu,
  - Hradlovanie napájania.

Hradlovanie hodinového signálu

Hradlovaním hodinového signálu sa zamedzí zmene stavu obvodov, ktoré nie sú potrebné pri výkone úlohy. Obvod odpojený od hodinového signálu (obrázok 5.3) nereaguje na zmeny signálov na vstupe a teda sa nemení jeho stav. Hodinový signál sa často hradluje počas prechodu dát cez prúd obvodov akým je napríklad prúd (Pipe-line) procesora.

Hradlovanie hodinového signálu obvodu je použiteľné ako rozhranie pre vyššie úrovne návrhu. Napríklad môže byť implementované pomocou riadiaceho signálu pripojeného na bit registra.

[2]Register-transfer level, pozri pojmi a skratky



Obr. 5.3: Odpojenie hodinového signálu v prúde obvodov

Zníženie frekvencie a napätia obvodu

Z rovnice dynamického výkonu 5.3 vyplýva, že dynamický výkon je kvadraticky závislí od napätia v obvode. Preto zníženie napätia v obvode má najväčší vplyv na zvyšovanie efektivity obvodu. Negatívnym dôsledkom znižovania napájacieho napätia je zvyšovanie oneskorenia obvodu. Oneskorenie obvodu môžeme vyjadriť pomocou rovnice 5.4.

$$\tau = kC_l \frac{V_{cc}}{(V_{cc} - V_t)^2} \tag{5.4}$$

Kde k je zosilňujúci faktor udávaný pre konkrétnu technológiu tranzistora, C_l je záťažová kapacita obvodu, V_{cc} je pracovné napätie a V_t je prahové napätie.

Oneskorenie obvodu udáva periódu počas, ktorej sa stav tranzistora mení a nie je v ustálenom stave. Podľa rovnice 5.4 je možné vypočítať maximálnu taktovaciu frekvenciu obvodu pomocou vzťahu medzi frekvenciou a periódou nasledovne:

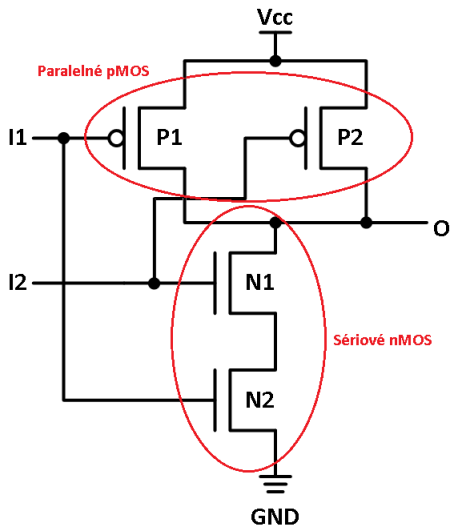
$$F_{max} = \frac{1}{T}, F_{max} = \frac{1}{kC_l} \frac{(V_{dd} - V_t)^2}{V_{dd}} \tag{5.5}$$

V prípade, že po znížení napájacieho napätia bude pôvodná frekvencia väčšia ako maximálna vypočítaná frekvencia F_{max} je nutné taktovať obvod nižšou frekvenciou. Potrebné zníženie frekvencie má za následok ďalšie zníženie dynamického výkonu na úkor predĺženia doby výpočtu.

Úsporný režim obvodu

Logické hradlo technológie CMOS pozostáva zo sériovo a paralelne zapojených tranzistorov (Obr. 5.4). Bez ohľadu na stav vstupov obvodu je vždy polovica tranzistorov vypnutá. Obvod má nižšiu spotrebu vtedy keď sú vypnuté sériovo zapojené tranzistory. Tomuto javu sa hovorí stohovací efekt (Stacking effect) [21].

Použitím tohto javu je možné počas kľudového stavu obvodu razantne znížiť únikový výkon v obvode. Výhodou tejto metódy je, že narozdiel od hradlovania napájacieho napätia

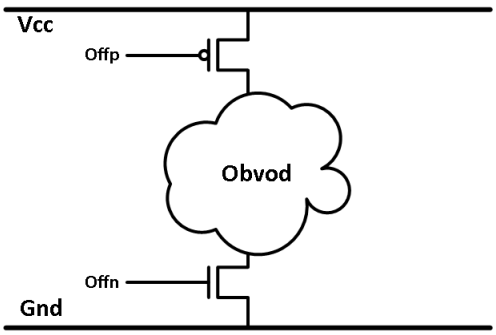


Obr. 5.4: Logické hradlo NAND

je obvod okamžite pripravený na použitie. V kombinácii s hradlovaním hodinového signálu je spravovaný obvod úspornejší aj z dynamickej aj zo statickej stránky.

Hradlovanie napájania

Najrazantnejším prístupom v znižovaní únikového výkonu je odpojenie nepoužívaných častí obvodu od napájacieho napätia. Na odpojenie obvodu (Obr. 5.5) sa používa tranzistor, ktorého kapacita je násobne väčšia ako preklápacia kapacita riadeného obvodu. Tranzistor musí mať väčšie rozmery aby dokázal zásobovať obvod dostatočným prúdom. Ak je tranzistor vypnutý obvod nemá žiaden únikový výkon. Na druhej strane tým, že tranzistor má vyššiu kapacitu jeho čas prepnutia stavu je dlhší ako u tranzistorov v obvode. Preto dôležitým parametrom obvodu je jeho veľkosť. Pri návrhu sa hľadá vhodná úroveň granularity obvodu.

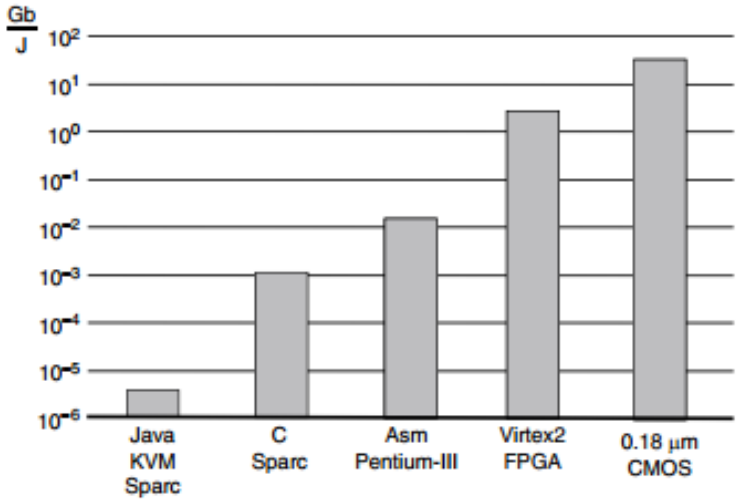


Obr. 5.5: Odpojenie napájania obvodu

5.4 Optimalizácia spotreby energie na úrovni Softvéru

Na softvérovej úrovni je možné použiť úsporné rozhrania z hardvéru. Tieto rozhrania môžu byť využité kompilátorom, operačným systémom alebo používateľom pri návrhu aplikačného softvéru. Okrem hardvérového rozhrania existujú optimalizačné metódy návrhu softvéru, ktoré prispievajú k zefektívňovaniu návrhu systému z pohľadu spotreby energie. V nasledujúcich kapitolách budú rozoberané viaceré prístupy.

S rastúcou abstraktnou úrovňou softvéru rastie aj spotreba systému. Obrázok 5.6 zobrazuje porovnanie počtu spracovaných bitov dát na jeden Jaul spotrebovanej energie rôznych úrovní abstrakcie.



Obr. 5.6: Energetická efektívnosť rôznych úrovní abstrakcie návrhu algoritmu AES [30].

Kompilátor

Úlohou kompilátora je prekladať navrhované používateľské aplikácie a operačný systém do strojového kódu, konkrétneho procesora. Preto je kompilátor dôležitým prvkom pri zvyšovaní energetickej efektivity systému. Pri kompilácií je možné optimalizovať množstvo prístupov do pamäte, veľkosť skompilovaného kódu. Kompilátor môže prispieť k úspore energie týmito metódami:

- Transformácia cyklov [13].
- Optimalizácie na inštrukčnej báze
- Optimalizácia prístupu do pamäte

Princípom transformácií cyklov je minimalizovanie množstva prístupov do pamäte. Minimalizáciou sa zabezpečí aj nižšia spotreba pamäte cache a hlavnej pamäte cite5-13n3.

Niektoré procesory sú vybavené dvoma inštrukčnými sadami. Základná sada obsahuje štandardný inštrukčný set a redukovaná sada redukovaný set inštrukcií, ktorá má menšiu šírku slova (štandardne polovičnú). Ak je procesor v móde s redukovanou inštrukčnou sadou, má k dispozícií menší počet registrov, využíva polovičnú pamäť programu a niektoré jeho funkčné jednotky nie sú aktívne. Preto celý procesor a aj systém spotrebuje menej energie [24].

Cieľom optimalizácie prístupu do pamäte je zmenšenie počtu prístupov (zápisov a čítaní) do pamäte. Tým sa zníži spotreba na zbernici ale aj v pamäti, kde dochádza k menšiemu počtu prepínaní tranzistorov. Počet prístupov sa dá zmenšiť napríklad ukladaním často používaných premenných priamo v registroch procesora.

Operačný systém

Príspevok operačného systému k zvyšovaniu energetickej efektivity môžeme rozdeliť do dvoch oblastí. Prvou oblasťou je efektívny návrh operačného systému, teda pasívny príspevok k efektivite [27]. Druhou oblasťou je kontrola a dynamická úprava spotreby celého systému, teda dynamický príspevok k efektivite. Z hľadiska dynamického príspevku operačný systém optimalizuje spotrebu energie [47]:

- Dynamickým škálovaním frekvencie a napätia
- Riadením spotreby prvkov systému

Z hľadiska dynamického škálovania frekvencie a napätia sa jedná o sledovanie utilizácie procesora a jeho periférií. Operačný systém vyhodnocuje stav systému ako celku a aplikuje podľa vybranej politiky úsporné opatrenia.

Povedzme, že procesor nemá žiadnu úlohu na spracovanie, v tom prípade operačný systém vyhodnotí, že je vhodné procesor uviesť do stavu s nízkou frekvenciou a napätím. Pri aplikovaní tejto metódy je dôležité zohľadniť čas potrebný na uvedenie systému do iného módu. Počas nízko výkonového stavu môže dôjsť k oneskorenému spracovaniu dôležitých prerušení, ktoré vedie k nedodržaniu doby odozvy systému.

Z hľadiska riadenia spotreby prvkov systému, operačný systém zabezpečuje správu zariadení. Používateľské aplikácie nebývajú navrhované s dôrazom na upratovanie si po sebe a preto toto upratovanie zabezpečuje operačný systém. Keď aplikácia prestane používať vybranú periférium operačný systém vyhodnotí jej ďalšie využitie. V prípade, že periféria nie je potrebná pre iné aplikácie operačný systém zabezpečí jej uvedenie do úsporného režimu. Výber typu úsporného režimu závisí od aktuálneho využitia periférie.

Napríklad majme periférne zariadenie, ktoré nie je zatiaľ používané žiadnou aplikáciou. V takomto prípade je periféria vypnutá úplne. V systéme môže byť spustená aplikácia,

ktorá požiada cez systémové volanie o pridelenie periférie na odoslanie dát a spätné prijatie potvrdenia. Operačný systém pripraví periférne zariadenie na odosielanie dát. Aplikácia odošle dáta a čaká na odpoveď. Operačný systém uvedie periférium do režimu spánku, v ktorom bude periféria čakať na potvrdzujúcu správu.

Aplikačný softvér

Na úrovni aplikačného softvéru je možné použiť rovnaké metódy ako v operačnom systéme a kompilátore. Nevýhodou aplikovania týchto metód v tejto úrovni je ich časová náročnosť. Na úrovni OS a kompilátora je problém vyriešený raz. V aplikačnom softvéri musí používateľ problematiku zvážiť znova. Zároveň používateľ musí mať skúsenosti s energetickou optimalizáciou.

5.5 Meranie spotreby systému

Prvotným odhadom spotreby systému ako celku je výpočet na základe dátových listov produktu. V dátových listoch je možné nájsť hodnoty spotreby produktu v rôznych módoch [46]. Katalógové údaje ale neposkytujú presný obraz o celkovej spotrebe systému pretože hodnoty v katalógu sú len orientačné a nezohľadňujú všetky možné prípady nastavenia systému.

Spotreba systému sa dá merať aplikáciou meracích zariadení priamo v prevádzke alebo počas testovania systému. Meracie zariadenie je pripojené na konkrétne časti systému alebo na systém ako celok. Výsledkom merania, ktoré prebieha počas celého cyklu systému, sú veľmi presné štatistiky, na základe, ktorých sa dajú hľadať možné optimalizácie systému.

Priame meranie nie je vhodné aplikovať na navrhované systémy pretože jeho značnými nevýhodami počas návrhu systému sú:

- Kúpa drahého meracieho zariadenia.
- Nutný výber hardvéru pred návrhom softvéru.
- Výsledky sú prístupné až po prebehnutí cyklu systému.
- Aplikovanie na navrhované systémy je nepraktické.
- Meranie možné až vo fáze testovania prototypu systému.

Kvôli takýmto nevýhodám je meranie spotreby neefektívne a predlžuje dobu návrhu systému. Preto je v takomto prípade spotrebu energie výhodné modelovať a simulovať.

5.6 Odhad spotreby systému

Model je zostavený na základe prvotného merania prvkov systému, ktoré môže dodávať výrobca hardvéru. Vývojár namiesto zdĺhavého merania odhaduje spotrebu na základe jednoduchšieho matematického modelu hardvéru. Odhad je prístupný prakticky okamžite a bez potreby zakúpenia drahého meracieho zariadenia a hardvéru. Návrh hardvéru môže prebiehať súčasne so softvérom.

Energetický model predstavuje možnosť presného odhadu spotreby energie vyvíjanej aplikácie bez potreby jej nasadenia na reálne fungujúci hardvér. Energetické modely systémov s procesorom sú tvorené rôznymi prístupmi. Z hľadiska granularity existujú modely založené na pozorovaní správania sa:

- **Logických obvodov** - Výhodou pozorovania správania sa logických obvodov procesora je presnosť vykonaného merania. Problémom modelu je, že veľké množstvo procesorov nemá zverejnený takýto opis a tým je nemožné takýto model vytvoriť [17].
- **Inštrukcií procesora** - Na rozdiel od predchádzajúceho modelu, Inštrukčný model nepotrebuje opis procesora v opisnom jazyku. Model je tvorený pomocou merania energie spotrebovanej počas vykonania konkrétnej inštrukcie. Procesor je braný ako čierna skrinka. Nevýhodou modelu je ale jeho výpočtová zložitosť [6][22][28][32][36].
- **Blokov systému** - Blokový model pracuje s procesorom ako so systémom funkčných blokov. Každý blok má v systéme svoju úlohu. Blok môže byť sériové rozhranie, analógovo digitálny prevodník, atď. V tomto modely procesor vystupuje ako šedá skrinka, teda je možné zohľadniť, z akých blokov sa procesor skladá [7].
- **Softvéru** - Na úrovni operačného systému je možné realizovať odhad spotreby energie použitím energetického modelu operačného systému. Operačný systém poskytuje používateľským aplikáciam volania, ktorých spotreba sa dá odmerať [5][12][23][33][45][48].

Z hľadiska analýzy spotreby systému poznáme:

- **Offline modely**, v ktorých sa analyzujú jednotlivé možné cesty prechodového grafu systému. Činitele, ktoré nie sú známe pred spustením sa berú do úvahy s priemerným alebo maximálnym dopadom na spotrebu.
- **Online modely**, v ktorých sa analyzuje aktuálny prechod grafom programu. Do úvahy sa môžu započítať aj činitele, ktoré neboli známe v offline fáze [5].

6 Tézy dizertačnej práce

V predchádzajúcich kapitolách sme načrtli problematiku súbežného návrhu hardvéru a softvéru. Identifikovali sme, že existujú dve vrstvy abstrakcie medzi skutočným hardvérom a jadrom operačného systému. Tieto vrstvi abstrakcie skrývajú zložitosť hardvéru do jednoduchšieho rozhrania a pamäťového priestoru. Komunikačným prostriedkom rozhrania sú inštrukcie jazyka symbolických inštrukcií, respektíve príkazy programovacieho jazyka. Tak tiež sme uviedli, že jadro operačného systému musí obísť vrstvu abstrakcie programovacieho jazyka nato, aby mohlo riadiť prepínanie úloh. Na túto činnosť využíva vrstvu abstrakcie strojového jazyka.

Identifikovali sme problém pri implementácií častí operačného systému, ktoré riadia a spravujú perifériu procesora. Procesor je obalený dvoma vrstvami abstrakcie, ktoré skrývajú vnútornú organizáciu procesora do pamäťového priestoru. Periféria procesora je sprístupnená pomocou adresácie. Návrhár operačného systému ale nemôže vedieť ako sa daná periféria správa a ako sa s ňou má komunikovať. Našou predstavou je toto správanie a komunikačné rozhranie formálne opísať vychádzajúc z dátového listu periférie. Výsledný formálny opis bude potom už len krok od automatizovaného generovania riadiaceho kódu pre danú periférium procesora, ktorý sa môže stať súčasťou jadra operačného systému.

V práci sme uviedli, že vrstvová a modulárna architektúra je vhodnou pre jadro operačného systému. Navrhujeme aby operačný systém pozostával z platformovo-závislej bázovej vrstvy a platformovo-nezávislej modulárnej vrstvy. Túto architektúru mienime podchytiť formálnym opisom.

Vychádzajúc z predchádzajúcej problematiky si kladieme nasledujúci cieľ dizertačnej práce:
Návrh formálneho opisu vnorených operačných systémov, ich komponentov a väzieb medzi komponentami v spojitosti s formálnym opisom procesorov a ich komponentov.
Čiastkové ciele:

- Návrh Modulárneho operačného systému pre vnorené aplikácie s použitím navrhnutého formálneho opisu vnorených operačných systémov.
- Implementácia Modulárneho operačného systémy na vybranej architektúre procesorov.
- Mapovanie formálneho opisu vnoreného operačného systému na existujúce a navrhované architektúry procesorov. S dôrazom na efektívne a energeticky efektívne využívanie potrebných hardvérových zdrojov systému.

- Zohľadnenie vnorených systémov s viacerými jadrami procesorov pri formálnom opise vnorených operačných systémov.
- Overenie správnosti navrhnutého formálneho opisu vnorených operačných systémov a porovnanie s inými postupmi návrhu vnorených operačných systémov.

7 Zhodnotenie

Vnorené operačné systémy sa rozšírili na široké spektrum vnorených systémov. Okrem štandardných vnorených systémov sú to napríklad vnorené systémy s rekonfigurovateľným hardvérom, viacerými jadrami procesorov, definovanou dobou odozvy, obmedzenými zdrojmi energie, bezdrôtovou architektúrou a mnohé iné. Pri všetkých týchto druhoch vnorených systémov sme identifikovali, že vrstvovo-modulárna architektúra vnorených operačných systémov je pri mapovaní na konkrétny hardvér tou správnou voľbou. Myšlienky vrstvovo-modulárnej architektúry sme prezentovali v dvoch publikáciach [41] [42].

Hardvér poskytuje množstvo mechanizmov, ktoré minimalizujú spotrebu energie avšak tieto mechanizmy pravdepodobne nie sú využívané aplikačnými používateľskými rozhraniami, knižnicami a operačnými systémami v plnej miere. Vychádzame z predpokladu, že v jednotlivých vrstvách abstrakcie medzi procesorom a jadrom operačného systému neexistuje formálne prepojenie s perifériou procesora.

Pre efektívne mapovanie softvéru na hardvér je vhodné použiť aparát formálneho opisu vnorených operačných systémov, ktorý umožní presné mapovanie systémových požiadaviek na vnorený operačný systém na existujúci alebo ešte len vznikajúci hardvér. Prínosom zavedenia formálneho opisu vnorených operačných systémov bude efektívne mapovanie softvéru na hardvér s využitím všetkých možností hardvéru.

Literatúra

[1] Al-Wattar, A.; Areibi, S.; Saffih, F.: Efficient On-line Hardware/Software Task Scheduling for Dynamic Run-time Reconfigurable Systems. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, may 2012, s. 401 –406.

[2] Atmel: *Datasheet, AT91SAM7S256*. 2010.

[3] Bailey, B.; Martin, G.: *ESL Models and their Application: Electronic System Level Design and Verification in Practice*. Springer, prvý vydanie, 2009, ISBN 1441909648, 9781441909640.

[4] Bogan, S.: *Formal Specification of a Simple Operating System*. Dizertačná práca, der Naturwissenschaftlich-Technischen Fakultäten der Universität des Saarlandes, 2008.

[5] Brandolese, C.; Corbetta, S.; Fornaciari, W.: Software energy estimation based on statistical characterization of intermediate compilation code. In *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, 2011, ISSN Pending, s. 333–338.

[6] Chattopadhyay, A.; Zhang, D.; Kammler, D.; aj.: Power-efficient Instruction Encoding Optimization for Embedded Processors. In *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, jan. 2007, ISSN 1063-9667, s. 595 –600.

[7] Dhouib, S.; Senn, E.; Diguet, J.-P.; aj.: Modelling and estimating the energy consumption of embedded applications and operating systems. In *Integrated Circuits, ISIC '09. Proceedings of the 2009 12th International Symposium on*, dec. 2009, s. 457 –461.

[8] Gargano, M.; Hillebrand, A., M.; Leinenbach, C., D.; aj.: On the correctness of operating system kernels. In *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005. Proceedings*, ročník 3603, 2005, ISSN 0302-9743, s. 1 –16.

[9] Gohringer, D.; Hubner, M.; Zeutebouo, E.; aj.: CAP-OS: Operating system for runtime scheduling, task mapping and resource management on reconfigurable multiprocessor architectures. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, april 2010, s. 1 –8.

[10] Gokhale, M. B.; Graham, P. S.: *Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays*. Springer Publishing Company, Incorporated, prvý vydanie, 2010, ISBN 1441938656, 9781441938657.

[11] Herman, J.; Kenna, C.; Mollison, M.; aj.: RTOS Support for Multicore Mixed-Criticality Systems. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, april 2012, ISSN 1080-1812, s. 197 –208.

[12] Hsu, C.-H.; Chen, J. J.; Tsao, S.-L.: Evaluation and modeling of power consumption of a heterogeneous dual-core processor. In *Parallel and Distributed Systems, 2007 International Conference on*, ročník 2, dec. 2007, ISSN 1521-9097, s. 1 –8.

[13] Kandemir, M.; Vijaykrishnan, N.; Irwin, M. J.; aj.: Influence of compiler optimizations on system power. *IEEE Trans. Very Large Scale Integr. Syst.*, ročník 9, č. 6, December 2001: s. 801–804, ISSN 1063-8210.
URL `http://dx.doi.org/10.1109/92.974893`

[14] Krill, B.; Amira, A.; Rabah, H.: Generic virtual filesystems for reconfigurable devices. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, may 2012, ISSN 0271-4302, s. 1815 –1818.

[15] Labrosse, J. J.; Ganssle, J.; Robert Oshana, e. a.: *Embedded Software: Know It All (Newnes Know It All)*. Newnes, 2007, ISBN 0750685832.

[16] Lee, E.; Messerschmitt, D.: Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. *Computers, IEEE Transactions on*, ročník C-36, č. 1, 1987: s. 24–35, ISSN 0018-9340.

[17] Marcon, C. A. M.; Filho, S. J.; Hessel, F. P.: A VHDL based approach for fast and accurate energy consumption estimations. In *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on*, oct. 2007, s. 276 –279.

[18] Marieska, M.; Hariyanto, P.; Fauzan, M.; aj.: On performance of kernel based and embedded Real-Time Operating System: Benchmarking and analysis. In *Advanced Computer Science and Information System (ICACSIS), 2011 International Conference on*, 2011, s. 401–406.

[19] Moore, G. E.: Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. *Solid-State Circuits Newsletter, IEEE*, ročník 11, č. 5, sept. 2006: s. 33 –35, ISSN 1098-4232.

[20] Myslík, J.: *Srovnávací studie vestavěných operačních systémů*. Bakalárska práca, České vysoké učení technické v Praze Fakulta elektrotechnická, may 2008.

[21] Nagar, A.; KUMAR, V. S.; P., K.: Power Minimization Of Logical Circuit Through Transistor Stacking. *International Journal of Innovative Technology and Research*, ročník 1, č. 3, 2013: str. 256 – 260.

[22] Nikolaidis, S.; Laopoulos, T.: Instruction-level power consumption estimation embedded processors low-power applications. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, International Workshop on, 2001.*, 2001, s. 139 –142.

[23] Ouni, B.; Belleudy, C.; Ben Rekhissa, H.; aj.: Energy leakage in low power embedded operating systems using DVFS policy. In *Faible Tension Faible Consommation (FTFC), 2012 IEEE*, june 2012, s. 1 –4.

[24] Panda, P.: *Power-Efficient System Design*. Springer, 2010, ISBN 9781441963888. URL `http://books.google.sk/books?id=uXyCqEJYITAC`

[25] Paul, R.; Saha, S.; Sau, S.; aj.: Real time communication between multiple FPGA systems in multitasking environment using RTOS. In *Devices, Circuits and Systems (ICDCS), 2012 International Conference on*, march 2012, s. 130 –134.

[26] Platzner, M.; Teich, J.; Wehn, N.: *Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications*. Springer Publishing Company, Incorporated, prvý vydanie, 2010, ISBN 9048134846, 9789048134847.

[27] Prathipati, R.: *Energy Efficient Scheduling Techniques for Real-time Embedded Systems*. Diplomová práca, Texas A and M University, 2004.

[28] Sami, M.; Sciuto, D.; Silvano, C.; aj.: An instruction-level energy model for embedded VLIW architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, ročník 21, č. 9, sep 2002: s. 998 – 1010, ISSN 0278-0070.

[29] Santambrogio, M.; Rana, V.; Sciuto, D.: Operating system support for online partial dynamic reconfiguration management. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, sept. 2008, s. 455 –458.

[30] Schaumont, P.: *A Practical Introduction to Hardware/Software Codesign.* Springer, 2010, ISBN 978-1-4419-5999-7.

[31] Steiger, C.; Walder, H.; Platzner, M.: Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks. *Computers, IEEE Transactions on*, ročník 53, č. 11, nov. 2004: s. 1393 – 1407, ISSN 0018-9340.

[32] Sultan, S.; Masud, S.: Rapid software power estimation of embedded pipelined processor through instruction level power model. In *Performance Evaluation of Computer Telecommunication Systems, 2009. SPECTS 2009. International Symposium on*, ročník 41, july 2009, s. 27 –34.

[33] Tan, T.; Raghunathan, A.; Jha, N.: Embedded operating system energy analysis and macro-modeling. In *Computer Design: VLSI in Computers and Processors, 2002. Proceedings. 2002 IEEE International Conference on*, 2002, ISSN 1063-6404, s. 515 – 522.

[34] Tanenbaum, A. S.: *Modern Operating Systems (2nd Edition) (GOAL Series)*. Prentice Hall, 2001, ISBN 0130313580.

[35] Tanenbaum, A. S.; Woodhull, A. S.: *Operating Systems Design and Implementation (3rd Edition)*. Prentice Hall, 2006, ISBN 0131429388.

[36] Tiwari, V.; Malik, S.; Wolfe, A.; aj.: Instruction level power analysis and optimization of software. In *VLSI Design, 1996. Proceedings., Ninth International Conference on*, jan 1996, ISSN 1063-9667, s. 326 –328.

[37] Tomiyama, H.; Honda, S.; Takada, H.: Real-time operating systems for multicore embedded systems. In *SoC Design Conference, 2008. ISOCC '08. International*, ročník 01, 2008, s. I–62–I–67.

[38] Štrba, A.: *Wireless Embedded System Powered by Energy Harvesting*. Dizertačná práca, Faculty of Informatics and Information Technologies, Máj 2011.

[39] Vijay, S.: *A Study of Real-Time Embedded Software Systems and Real-time Operating Systems*. Diplomová práca, Kanwal Rekhi School of Information Technology Indian Institute of Technology, Bombay Mumbai, 2002.

[40] Vojtko, M.: *Modulárny operačný systém pre vnorené systémy*. Diplomová práca, Faculty of Informatics and Information Technologies, 2012.

[41] Vojtko, M.: Modular Operating system. In *Student Research Conference 2013, Vol. 2 : 9th Student Research Conference in Informatics and Information Technologies Bratislava*, ročník 9, 2013, ISBN 978-80-227-3911-5, s. 283–290.

[42] Vojtko, M.; Krajčovič, T.: Prototype of Modular Operating System for Embedded Applications. In *Applied Electronics - 2013 International Conference on Applied Electronics*, ročník 18, 2013, ISBN 978-80-261-0166-6, ISSN 1803-7232, s. 317–320.

[43] Voros, N.; Masselos, K.: *System Level Design of Reconfigurable Systems-on-Chip*. Springer, 2005, ISBN 9780387261034.
URL `http://books.google.sk/books?id=nw7g6QvLU1IC`

[44] Wang, C.; Yao, B.; Yang, Y.; aj.: A Survey of Embedded Operating System, 2001, nepublikované.

[45] Wang, Q.; Yang, W.: Energy Consumption Model for Power Management in Wireless Sensor Networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON '07. 4th Annual IEEE Communications Society Conference on*, june 2007, s. 142 –151.

[46] Wolf, W.: Household hints for embedded systems designers. *Computer*, ročník 35, č. 5, may 2002: s. 106 –108, ISSN 0018-9162.

[47] Y.H., Y.: *Ultra Low Power Application Specific Instruction-set Processor Design, for a cardiac beat detector algorithm*. Diplomová práca, Norwegian University of Science and Technology, 2009.

[48] Zhao, X.; Guo, Y.; Wang, H.; aj.: Fine-Grained Energy Estimation and Optimization of Embedded Operating Systems. In *Embedded Software and Systems Symposia, 2008. ICESS Symposia '08. International Conference on*, july 2008, s. 90 –95.

A Zoznam publikácií autora

1. Vojtko, Martin - Krajčovič, Tibor: Prototype of Modular Operating System for Embedded App-lications. In: Applied Electronics 2013 : Interna-tional Conference on Applied Electronics. Pilsen, Czech Republic, 9-11 September 2013. - Plzeň : University of West Bohemia, 2013. - ISBN 978-80-261-0166-6. - S. 317-320

   [vnútrofakult. kateg.: B]

2. Vojtko, Martin: Modular Operating System. In: Student Research Conference 2013, Vol. 2 : 9th Student Research Conference in Informatics and Information Technologies Bratislava, May 4, 2013. Proceedings. Vol. 2. - Bratislava : Nakladateľstvo STU, 2013. - ISBN 978-80-2227-3911-5. - S. 283-290

   [vnútrofakult. kateg.: D]

3. Vojtko, Martin: Modulárny operačný systém pre vnorené systémy. In: PAD 2012 - Počítačové architektury a diagnostika : Pracovní seminář pro studenty doktorského studia. Milovy ve Žďárských vrších, Czech republic, 10.-12.9.2012. - Praha : Česká technika-nakl.ČVUT, 2012. - ISBN 978-80-01-05106-1. - S. 23-27

   [vnútrofakult. kateg.: D]

4. Babiš, Miroslav - Ďuríček, Maroš - Harvanová, Valéria - Vojtko, Martin: Forest Guardian - monitoring System for Detecting Logging Activities Based on Sound Recognition. In: Student Research Conference 2011, Vol. 2 : 7th Student Research Conference in Informatics and Information Technologies Bratislava, May 4, 2011. Proceedings. Vol. 2. - Bratislava : Nakladateľstvo STU, 2011. - ISBN 978-80-227-3488-2. - S. 354-359

   [vnútrofakult. kateg.: D]