VUT v Brně
Fakulta informačních technologií

Rekonfigurovatelné výpočetní platformy

Habilitační práce

Obor habilitace: **Výpočetní technika a informatika**
Uchazeč: **Dr. Ing. Otto Fučík**

Brno, prosinec 2008

ABSTRAKT

Práce se zabývá pokročilými technikami návrhu aplikačně specifických výpočetních systémů, které se snaží o optimalizaci často protichůdných kritérií kladených na jejich implementaci v rámci daném dostupnými technologiemi a společenskou poptávku. Autor se především zaměřil na výzkum, návrh a použití výpočetních platforem s programovatelným hardwarem při tvorbě inženýrských děl s důrazem na techniky souběžného návrhu hardware a software jak vestavěných systémů, tak komplexních vizuálních systémů.

S ohledem na složitost a heterogenní podstatu dnešních výpočetních platforem a systémů je v práci prezentována snaha o jejich modelování na co nejvyšší úrovni abstrakce takovými výpočetními modely, které nejlépe zohledňují charakter dané úlohy a usnadňují tak její efektivní implementaci. Problematika je rozvedena s důrazem na modelování času a použití tzv. globálně asynchronních a lokálně synchronních obvodů. Dále je diskutována problematika souběžného návrhu hardware a software výpočetních systémů s ohledem na optimalizaci výsledného řešení s důrazem na snížení příkonu výsledného systému. Je též zmíněna důležitost specifikace a validace výsledného produktu.

V práci je prezentována řada vysoce výkonných výpočetních platforem využívajících reprogramovatelný hardware (FPGA), které byly využity ve výzkumu, výuce a v komerčních aplikacích. Další oblastí, ve které autor aktivně pracoval a je v práci popsána, je tvorba komplexních vizuálních systémů pro dopravní a průmyslové aplikace s velkým společenským dopadem.

ABSTRACT

The presented work deals with advanced techniques for the design of application-specific computing systems, which are attempting to optimise contradictory criteria based on accessible technologies and social demand. The author's priority has been focused on the research, design and exploitation of computing platforms with programmable hardware during the creation of engineering tasks. A stress has been laid on the technique of collateral design of hardware and software of both embedded and complex visual systems.

Because nowadays computational platforms are complex and heterogeneous the work presents a modelling attempt on high level of abstraction. The selected computational models should as much as possible take into account the character of a given task as well as its effective implementation. The problematic is detailed with the stress on time modelling and application of so called globally-asynchronous and locally synchronous circuits. Further to this a problematic of collateral design and hardware and software of computational systems with the view of optimisation of the final solution with the emphasis on reduction of power requirement of the resulting system is also discussed. The importance of the specification and validation of the final product has been mentioned as well.

The work presents a number of highly performing computational platforms based on reprogrammable hardware (FPGA). They have been used in the research, teaching and in many commercial applications. Further areas covered actively by the author and described in the work are development of complex visual systems for transport and industrial applications with large societal impact.

1 ÚVOD

Úspěšný produkt je výsledkem návrhu, který zohledňuje kompromisy mezi potřebami uživatelů, požadovanými vlastnostmi (cena, výkonnost, spotřeba energie atd.) a prostředím, ve kterém se bude používat (životní prostředí, sociální, politické, etické, zdravotní, bezpečnostní, výrobní, servisní a další aspekty). Vývoj složitých výpočetních systémů je komplexní činnost, která vyžaduje týmovou práci a musí probíhat v těsné spolupráci s uživatelem. Jde o spojení vědeckých metod, inženýrského přístupu (technické vzdělání, zkušenosti) a umění (intuice, invence) v rámci daném technologickými možnostmi a stavem poznání. Při tvorbě informačních systémů, je tradičně kladen důraz především na efektivitu procesu jejich návrhu (programovací jazyky, kompilátory, operační systémy, knihovní funkce) než efektivitou výsledné aplikace (výkonnost, rozměry, příkon, atd.). Je to dáno tím, že cena, která je dnes prakticky nejdůležitějším kritériem, je z převážné míry tvořena cenou lidské práce a okamžikem uvedení systému na trh. S rostoucí poptávkou po výpočetně náročných, energeticky úsporných a často přenosných zařízení je však třeba používat takové návrhové techniky, které kladou důraz jak na efektivitu návrhu, tak výsledného systému.

Tato práce se si klade za cíl výzkum a praktické použití pokročilých technik návrhu aplikačně specifických výpočetních systémů, které umožňují optimalizaci často protichůdných kritérií kladených na jejich implementaci v rámci daném dostupnými technologiemi a společenskou poptávku. Práce je podána formou souboru významných inženýrských děl s velkým společenským dopadem a ohlasy, který je doplněn komentářem. S ohledem na značný rozsah dokumentace k jednotlivým systémům, jsou jejich hlavní charakteristiky prezentovány ve formě publikací v mezinárodních časopisech a na mezinárodních konferencích (viz vybrané publikace [22], [76], [73], [46], [74], [75], [10] a [4], jejichž kopie jsou v příloze). Díky složitosti jsou prezentovaná inženýrské díla[1] kolektivními pracemi, na kterých se různým způsobem podílely desítky spolupracovníků. Autor zde figuroval jako jeden z iniciátorů myšlenky, spolutvůrce koncepce a architektury jednotlivých systémů a výpočetních platforem a v řadě případů jako vedoucí realizačního týmu. Dále pak je spoluautorem návrhu a implementace hardwarových subsystémů (inteligentní kamery, výpočetní jednotky atd.).

Zaměření jednotlivých kapitol práce je následující: Po úvodu následuje druhá kapitola, která přibližuje problematiku souběžného návrhu software a hardware výpočetních systémů (HSC) s využitím rekonfigurovatelných výpočetních platforem.

V dnešní době nabývá problematika HSC na důležitosti, neboť technologie výroby integrovaných obvodů umožňují integrovat na jedné polovodičové podložce obrovské množství tranzistorů, které je třeba umět efektivně využít. Možnosti technologie FPGA (angl. Field Programmable Gate Array – FPGA) která, díky svým vlastnostem otevírá nové možnosti rychlé tvorby efektivních (cena, výkonnost, příkon apod.) výpočetních systémů, ve kterých není třeba HW vyrábět, ale stačí je jen (spolu se SW) naprogramovat jsou diskutovány v kapitole 3 .

V kapitole 4 jsou představeny různé výpočetní platformy s obvody FPGA.

S ohledem na složitost a heterogennost dnešních výpočetních platforem a systémů je třeba je modelovat na co nejvyšší úrovni abstrakce takovými výpočetními modely, které nejlépe zohledňují charakter dané úlohy a usnadňují tak její efektivní implementaci. V kapitole 5 je tato problematika rozvedena s důrazem na modelování času a použití tzv. globálně asynchronních a lokálně synchronních obvodů (GALS).

HSC návrh výpočetních systémů je charakteristický kvantitativním přístupem. Pro optimalizaci rozdělení systému mezi části vykonávané různým způsobem v SW a HW je nezbytná jak analýza

[1] Většina prezentovaných děl byla vytvořena za finančního přispění firmy Camea, spol. s r o. V případě financování z jiných zdrojů jsou v textu a publikacích uvedeny příslušné odkazy.

(evaluace, odhady) vlastností jednotlivých částí a jejich interakci, tak chování výsledného systému. Uvedené problematice se věnuje kapitola 6 .

Důležitým krokem HSC výpočetních systémů je přidělení dostupných výpočetních prostředků jednotlivým výpočetním úlohám dané aplikace. Proces dekompozice struktury a chování systému a přiřazení komponent úlohám je předmětem procesu, který nazýváme „rozdělování“ (angl. partitioning). Uvedená problematika je přiblížena v kapitole 7 .

Validace je proces, při kterém se zjišťuje, zda produkt či krok při návrhu odpovídá svému účelu, splňuje veškerá daná omezení (cena, výkonnost, příkon ap.) a bude pracovat, jak je vyžadováno, že je tedy implementován správný produkt, viz diskuze v kapitole 8 .

Konečně návrhu komplexních systémů musí předcházet jejich specifikace a návrh architektury. Tato problematika je uvedena v kapitole 9 .

Kapitola 12 je přílohou, ve které uveden popis rodiny komplexních vizuálních systémů s velkým společenským dopadem, které byly navrhovány za použití HSC technik a jako integrující prvek využívají rekonfigurovatelné výpočetní platformy.

V kapitole 13 jsou kopie publikací popisující hlavní charakteristiky významných inženýrských děl s velkým společenským dopadem, které představují těžiště práce[2] a jsou podobněji komentovány v textu.

[2] Podrobnou dokumentaci k jednotlivým inženýrským dílům je možno vyžádat u autora.

2 SOUBĚŽNÝ NÁVRH

2.1 MOTIVACE

Gordon Moore (spoluzakladatel firmy Intel) formuloval v roce 1965 dodnes platný tzv. Moorův zákon [45], který říká, že stupeň integrace integrovaných obvodů, při které mají tranzistory nejnižší cenu, se bude zhruba zdvojnásobovat každého cca 1,5 roku. Problémem je však jejich efektivní využití, neboť na základě zjištění Freda Pollacka (firma Intel) se ukazuje, že efektivní (užitečná) výkonnost mikroprocesorů roste úměrně se čtvercem jejich složitosti[3] [55] (pozn.: příkon však roste lineárně). Složitost integrovaných obvodů tedy roste o cca 60 % ročně, avšak produktivita práce při návrhu výpočetních systémů jen o cca 20 % [33]. Dále se ukazuje, že frekvenci, na které pracují procesory, bude do budoucna obtížné zvyšovat stejným tempem jako doposud [52].

Řešením problematiky efektivního návrhu složitých výpočetních systémů v rámci daném jak uvedenými fyzikálními omezeními, tak často protichůdnými požadavky na jejich vlastnosti, se zabývá metodika tzv. souběžného návrhu anglicky nazývaná „Combined/Cooperative/Concurrent Design“, zkráceně „Co-design/Codesign“. Problematika se typicky zužuje na souběžný návrh hardware (HW) a software (SW) – „HW/SW Codesign – HSC“. Techniky HSC se tradičně věnují především návrhu vestavěných systémů (angl. Embedded Systems – ES), což jsou informační systémy, které jsou vestavěny do větších produktů. Orientace HSC metodik na návrh ES je dána jejich postavením na trhu s elektronikou. V roce 2002 bylo vyrobeno cca 60 miliónů tranzistorů na každého člověka na planetě a v roce 2010 to bude cca 1 miliarda [71]. Roční obrat je dnes více než 300 miliard dolarů. Procesory všech typů, od 4bitových, přes DSP až po procesory pro osobní počítače (angl. Personal Computer – PC) a příslušné periferní obvody mají cca 30% podíl na trhu s číslicovými integrovanými obvody (IO) a generují zisk cca 60 miliard dolarů na globálním elektronickém trhu. Z toho procesory pro PC představují jen 2 % a zbytek je určen pro vestavěné systémy. Nejvíce se prodávají 8bitové procesory (nejvíce 8051 a 6805) s obratem cca 3 miliardy dolarů za rok (15 % zisku) [65].

2.2 TRADIČNÍ A SOUBĚŽNÝ NÁVRH

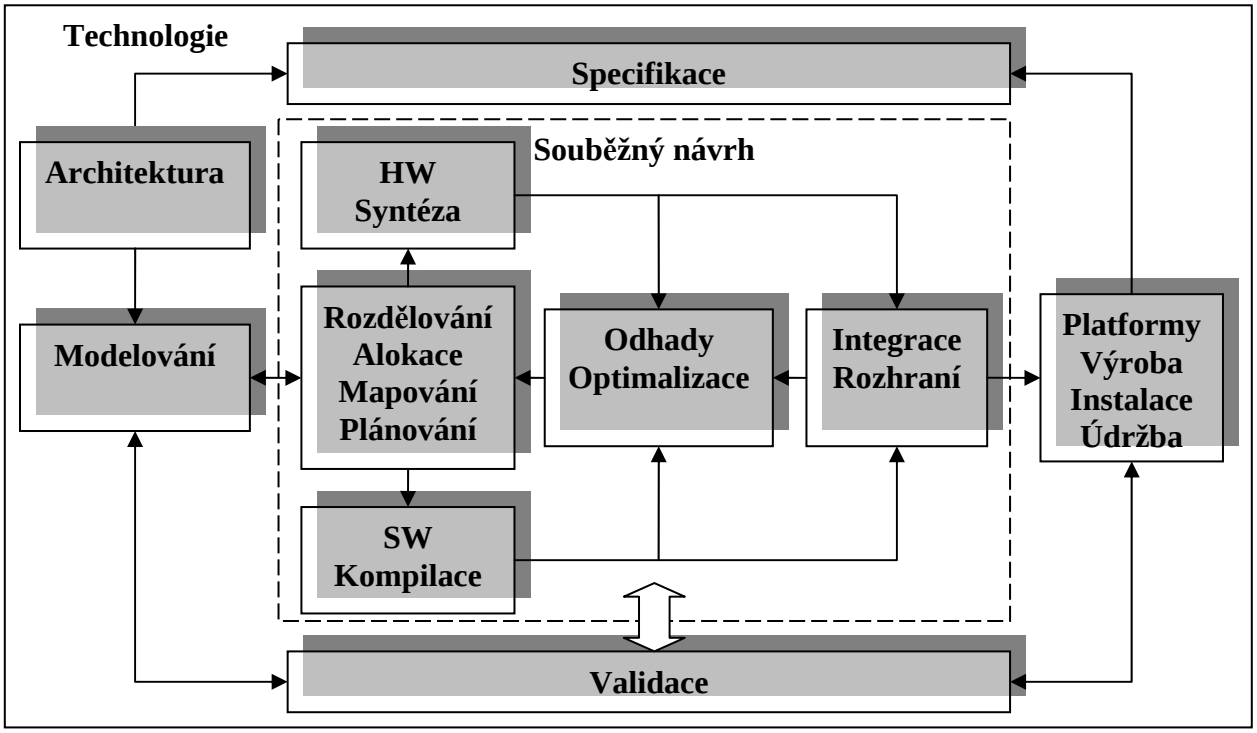
Tradiční návrh výpočetních systémů probíhá typicky tak, že se nejprve vyrobí HW systému, pro který se následně tvoří SW. Nevýhodou je, že sčítají doby návrhu a aplikace má vlastnosti dané především vlastnostmi použitého HW. Oproti tomu HSC metodika se snaží o tvorbu HW a SW pokud možno souběžně. Tímto je možno dosáhnout lepších vlastností výsledného systému, neboť lze optimalizovat využití výpočetních prostředků, díky úzké [34] kooperaci mezi návrhem SW a HW a bezprostřední validaci jejich vzájemných interakcí. Rychlost, s jakou je uveden produkt na trh, určuje podstatnou položkou ceny produktu a právě souběžný návrh ji může výrazně zkrátit. Uvedený postup, jednotlivé činnosti a vazby mezi nimi jsou ilustrovány na Obr. 1.

V této práci jsou prezentovány zkušenosti s aplikací HSC technik jak na úrovni výpočetních platforem (kapitola 4), tak s jejich zobecněním pro tvorbu komplexních výpočetních systémů (viz kapitola 12). Důraz je kladen na maximální opětovné použití v praxi osvědčených komponent a flexibilitu nejen SW, ale též HW díky použití programovatelného hardware.

Techniky HSC byly v různých formách používány již od počátků rozvoje výpočetní techniky, kdy se návrh HW počítačů prováděl souběžně s jejich programováním (viz např. první počítače [34]). Později se však procesy tvorby SW od návrhu HW často oddělovaly především proto, že na univerzálním počítači lze vykonávat libovolný program, což s sebou nese možnost využití jedné architektury počítače velkou skupinou programátorů, kteří se soustřeďují na tvorbu SW aplikací a nezdržují se při tom návrhem a výrobou HW. Pokrok v technologiích výroby integrovaných

[3]

obvodů s velkou hustotou integrace (angl. Very Large Scale Integration – VLSI) umožnil výrobu aplikačně specifických IO, které jsou navrhovány a vyráběny pro konkrétního zákazníka (angl. Application Specific Integration Circuit – ASIC) a jednočipových procesorů (angl. Microprocessor Unit – MPU). Především díky snadnému použití MPU, kdy hlavní činností při návrhu systému je programování, se otevřela cesta k rychlé tvorbě vestavěných systémů. MPU se průběžně doplňovaly o další obvody, které vylepšují jejich vlastnosti, především s ohledem na výpočetní výkon (např. ASIC koprocesor pro urychlení výpočtu). U takových systémů je tedy potřeba nejen programovat SW a navrhovat HW výpočetního systému, ale též optimalizovat rozdělení (angl. partitioning) výpočtu aplikace mezi SW (běžící na MPU) a přídavný HW (např. koprocesor). Pozn.: pravděpodobně první zmínka o HSC tak, jak jej chápeme dnes je z roku 1985 [60].

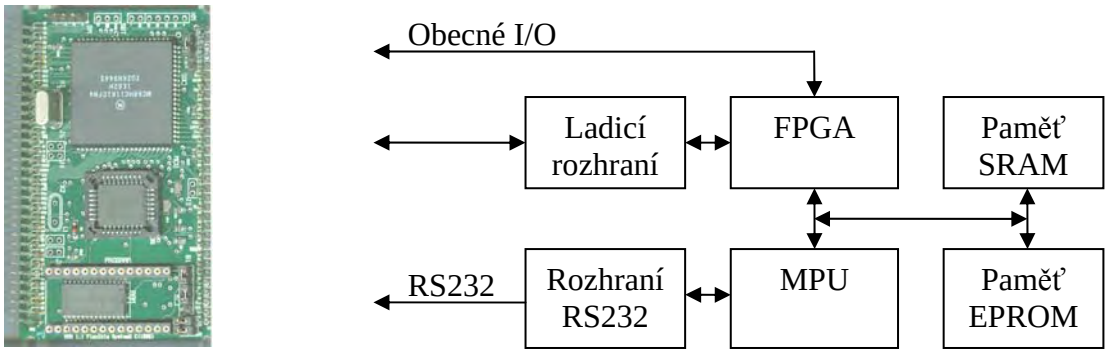


Obr. 1 HW/SW Codesign

Zpočátku se jevil jako dosažitelný cíl HSC technik návrhu vytvoření unifikované reprezentace systému univerzálním modelem a specifikačním jazykem [56]. Hlavní myšlenkou bylo vyjít z jednotné specifikace a postupnými návrhovými kroky ji zjemňovat, od čistě behaviorálního popisu na nejvyšší úrovni abstrakce, přes strukturní popis, až po implementaci a generování výrobních podkladů. Dále se předpokládalo, že vlastní proces, který rozhodne o tom, kde je slabé místo softwarového řešení, izoluje jej a implementuje v hardware, lze plně automatizovat. Výzkum tak byl převážně zaměřen na tvorbu nástrojů pro automatizovaný návrh. Typickým předpokladem bylo, že SW má běžet na nějakém procesoru a HW se bude používat pro akceleraci výpočtů, jejichž zpracování vyžaduje výpočetní výkon, který nelze daným SW řešením dosáhnout. Technické řešení HSC systémů se vycházelo z tehdejších technologických možností a jednalo se typicky o spojení MPU s dalším HW (např. ASIC) na deskách s plošnými spoji (angl. Printed Circuit Board – PCB), kdy SW se většinou programoval v assembleru a HW se specifikoval pomocí schémat.

Příkladem typického HSC systému 90. let 20. století je platforma MMX, která byla vyvinuta v roce 1994 [23] a [61]. Na desce je MPU, programovatelné logický obvod FPGA, přídavné paměti a komunikační rozhraní. Výpočetní jednotky realizované v FPGA jsou mapovány do

paměťového prostoru MPU a pracují jako koprocesory. Blokové schéma a fotografie systému MMX je na Obr. 2.

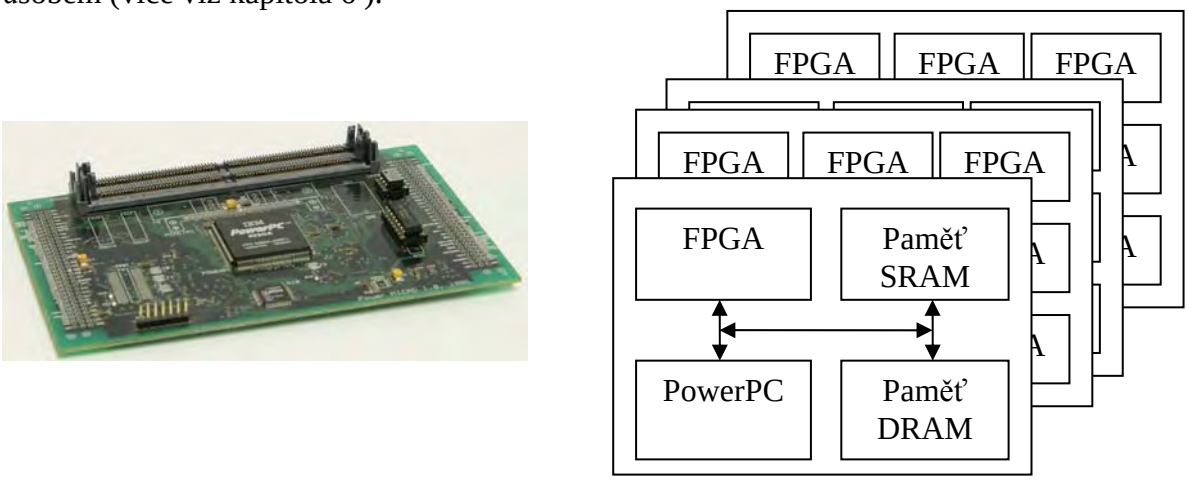


Obr. 2 Systém MMX

Systém MMX má též podporu tzv. dynamické rekonfigurace FPGA (více viz odstavec 3.5). HSC koncept systému MMX byl využit pro výzkum [17], výuku a tvorbu diplomových projektů na několika technických univerzitách (ÚIVT FEI VUT v Brně, University of Wyoming, USA) a využit v řadě komerčních aplikací.

Příkladem dalšího HSC systému realizovaného v té době je velmi výkonná výpočetní architektura FLEDGE, která byla realizována v roce 1995[4] [16]. Základní deska systému je osazena 32 bitovým MPU (PowerPC), obvodem FPGA, 256 MB paměti DRAM a 128 MB paměti SRAM. Ke každé základní desce lze připojit až 8 přídavných desek, z nichž každá nese až 8 obvodů FPGA. Základní desky mohou dále rychle sériově komunikovat každý až se šesti sousedními uzly, což umožňuje tvorbu víceprocesorových systémů. Blokové schéma a fotografie systému MMX je na Obr. 3.

Pro zkoumání možností tvorby systémů pomocí HSC byly navrženy desítky výpočetních jednotek, jako jsou akcelerátory výpočtů, rozhraní, číslicové filtry, matematické operace ap., které byly využívány, jak při výzkumu a ve výuce návrhu HW, tak v komerčních aplikacích. Pokud mají funkční jednotky charakter původních autorských návrhů, budeme je v dalším textu nazývat IPC (z anglického Intellectual Property Core). Na základě zkušeností lze konstatovat, že využití předem navržených a validovaných funkčních jednotek se může účinně přispět k řešení problémů spojených se složitostí návrhu, kdy návrhář konstruuje aplikaci ze stavebních prvků hierarchickým způsobem (více viz kapitola 6).



Obr. 3 Systém FLEDGE

[4] Systém FLEDGE byl vytvořen za finanční podpory grantu GAČR 102/95/1334.

2.3 ČINNOSTI

HSC návrhu výpočetních systémů zahrnuje řadu činností. Pro ilustraci si zde uveďme jejich heslovitý výčet s tím, že budou postupně podrobněji diskutovány v rámci dalších kapitol:

Specifikace
- Průzkum trhu (co na trhu chybí)
- Potřeby (co zákazník potřebuje)
- Charakteristiky systému (co je požadováno)
- Požadavky na systém (jak toho bude dosaženo)
- Kvantitativní analýza (studie proveditelnosti)
- Vlastnosti systému (bezpečnost, zabezpečení, zodpovědnosti)
- Management (lidské zdroje, koordinace, řízení projektů, outsourcing)
- Obchod (marketing, finance, outsourcing)

Architektura
- Výpočet (propustnost, latence)
- Data (typ, objem)
- Komunikace (zasílání zpráv, sdílená paměť)
- Rozhraní (sběrnice, sériové atd.)
- Protokoly (TCP/IP atd.)
- Synchronizace (handshake, semafor, atd.)
- Řízení
- Fyzikální vlastnosti (rozměry, příkon)

Modelování
- Výpočetní modely
- Komunikační modely

Souběžný návrh
- Rozdělování
  - Alokace (výběr výpočetních prostředků)
  - Mapování (přidělení výpočtů výpočetním prostředkům)
  - Plánování (sdílení výpočetních prostředků mezi výpočty)
- Transformace (syntéza HW, kompilace SW)
- Optimalizace (příkon, výkonnost, cena)
- Odhady (složitost, rychlost, paměť, příkon)
- Integrace (rozhraní, komunikace)
- Rozhraní (protokol, médium)
- Metodika (postup shora-dolů, zdola-nahoru)
- Metody (adaptivní, iterativní)
- Nástroje (syntéza, kompilace, framework)

Platformy
Prototyp (pilotní provoz)
Výroba (výrobní podklady, logistika)
Instalace
Údržba (update, upgrade)
Validace (verifikace, simulace, testování, emulace, rychlé prototypování, certifikace, kvalita)

Uvedený výčet činností vyplývá ze zkušeností z HSC návrhu řady výpočetních systémů pro zpracování multimediálních dat v reálném čase a komplexních vizuálních systémů pro dopravní a průmyslové aplikace viz kapitoly 12 a 13 . Prezentovaný pohled na danou problematiku je

rozšířením tradičních HSC metodik, které se často soustřeďují jen jistou část životního cyklu produktu (typicky jen fázi návrh), na systémovou úroveň (viz kapitola 9). Dalším aspektem je též důraz na adaptivní metody návrhu a bezprostřední validaci jednotlivých iterací návrhu.

2.4 METODIKA

Častý postup při vývoji jednoduchých systémů metodou „vytvoř a oprav“ je dnes, díky enormně rostoucí složitosti aplikací, nepoužitelný. Řešením je formalizace návrhových postupů s cílem zajistit, aby proces návrhu byl predikovatelný a efektivní. S ohledem na charakter výsledného produktu je doba strávená při jeho návrhu a výrobě různá. Např. při stavbě domu má projekt hodnotu cca 5% ceny domu a na jeho tvorbě se stráví též kolem 5% celkové doby stavby. Výroba (stavba) tedy zabírá podstatnou část jak ceny, tak doby realizace. Plány domu by tedy měly být neměnné po celou dobu stavby, neboť jakékoliv jejich úpravy při realizaci jsou velmi drahé. V případě výpočetních systémů je tomu naopak, návrh zabírá dominantní část trvání projektu a tedy i ceny produktu. Dnes se, díky využití výpočetních platforem (viz kapitola 4), výroba výpočetních systémů zjednodušuje prakticky na kopírování (program SW, konfigurace HW). Dále platí, že změna požadavků je při tvorbě výpočetních systémů běžným stavem. Praxe ukazuje, že dopředu nelze vše přesně naplánovat a specifikovat. A i kdyby se to podařilo, pak nelze předpokládat, že specifikace je bez chyby, které lze často odhalit např. až při testování prototypu a nebude ji tedy třeba měnit.

Na tuto skutečnost reagují adaptivní (inkrementální, evoluční) [70] návrhové metody (v oblasti softwarového inženýrství známé též např. jako agilní návrhové metody [14]), které kladou důraz na vlastní proces návrhu a ne na vytvoření detailního plánu. U adaptivního návrhu jsou jednotlivé iterace prováděny v krátkých časových intervalech. Návrh probíhá v iteracích, se zavedením zpětné vazby, kterou je ovlivněna tvorba na základě verifikace, validace a testování po každé iteraci. V každé iteraci se provádí návrh relativně malých částí systému, které lze plně validovat a ze kterých se potom lépe integruje celý systém. Předem formulovaná specifikace a architektura (viz kapitola 9.2) systému představuje dlouhodobý plán určující rámec, ve kterém se návrh realizuje (cena, doba návrhu), ale z hlediska jednotlivých kroků při návrhu nejsou fixní. Krátkodobé plány pak definují požadavky na návrh v dané iteraci. Řada návrhů komplexních systémů je prováděna na zakázku, kdy se nejčastěji vychází z pevné ceny a doby, za kterou se má produkt vyrobit a případně se mění rozsah, který bude projekt mít. Na základě zkušeností s adaptivním přístupem k návrhu systémů lze konstatovat, že orientace na potřeby zákazníka a tvůrčího týmu, avšak ne plány a jejich striktní dodržování, umožňuje dosáhnout lepších užitných vlastností produktu v kratším čase.

Díky důslednému používání programovatelných HW platforem (viz kapitola 4) se při tvorbě rozsáhlých aplikací (viz kapitoly 12 a 13) otevřela možnost aplikovat adaptivní metodiky návrhu na systémovou úroveň návrhu HSC. Z tohoto pohledu se do značné míry podařilo setřít rozdíly mezi návrhem SW a HW.

2.5 POSTUP

Složité systémy jsou sestaveny z jednodušších komponent, které vzájemně komunikují a kooperují a společně utváří chování či vlastnost systému, jež jde nad rámec možností jednotlivých komponent. Návrh musí respektovat omezení (angl. constraints), která jsou dána požadovanými vlastnostmi produktu (rozměry, příkon, propustnost dat atd.) a možnostmi jeho návrhu (doba uvedení na trh, cena atd.).

Při tvorbě složitých systémů je třeba zavádět abstrakci a hierarchické uspořádání, ve kterém je na každá úrovni omezena složitost (snáze pochopitelné, přehledné) a umožňuje opětovné použití komponent. Rozhraní mezi stavebními bloky představuje hlavní mechanizmus pro zavedení abstrakce. Rozhraní má často delší životnost než technologie, které propojuje (např. rozhraní

standardu RS232 standardizované v roce 1960 se používá dodnes a stále představuje jedno z nejvíce používaných). Rozhraní izoluje různé technologie od sebe a tím umožňuje jejich inovace a tím i evoluci systému s využitím nově dostupných technologií. Příkladem nejsou jen periferní zařízení, ale též rozhraní mezi SW a HW. Nejmarkantnějším příkladem je instrukční sada (angl. Instruction Set Architecture – ISA) procesorů firmy Intel, která byla navržena již v roce 1974 a je, vzhledem na potřebu zpětné kompatibility, dodnes standardem.

Pokud budeme navrhovat produkt, jehož žádná část ještě neexistuje, pak lze teoreticky provést jeho optimální syntézu. Popis chování systému lze vhodným způsobem modelovat s použitím vhodné abstrakce, která nejlépe postihuje charakter řešené úlohy. Takovému procesu návrhu říkáme shora-dolů (angl top-down). V praxi však použití čistě tohoto postupu spíše výjimkou. Z ekonomických důvodů a s ohledem na složitost bývá výsledný produkt částečně složen z již existujících komponent. Zde hovoříme o procesu návrhu zdola-nahoru (angl. bottom-up). Výhodou je, že nemusíme vše navrhovat od počátku, ale využíváme předem ověřené komponenty, u kterých lze přesně evaluovat, jak fungují a jaké mají vlastnosti (výkonnost, příkon, potřebná kapacita paměti atd.). Komponenty použité při návrhu však mohou vnést nějakou režii (nejsou třeba plně využity, je třeba je propojit pomocí rozhraní atd.), která se promítne do výsledného produktu.

Metodika HSC, která byla ověřena při tvorbě komplexních výpočetních systémů a jejich komponent představuje kombinaci přístupů shora-dolů a zdola-nahoru. Umožňuje využít ověřené komponenty, zohlednit možnosti technologie, kterou máme k dispozici a přitom navrhovat na co nejvyšší úrovni abstrakce.

3 TECHNOLOGIE

Jedním z klíčových parametrů, který umožňuje stavbu složitých výpočetních systémů, je spolehlivost použitých komponent. Teprve úspěšná výroba integrovaných obvodů umožnila integraci velkého množství tranzistorů na jedné polovodičové podložce a tím tvorbu komplexních výpočetních systémů s vysokou spolehlivostí. Díky technologii výroby integrovaných obvodů je též možno realizovat spolehlivé paměti s dostatečnou kapacitou a rychlostí, což fakticky otevřelo prostor pro rozvoj počítačů. Vlastnosti pamětí fakticky určují limity, kterých lze při aplikaci výpočetních strojů dosáhnout.

Při návrhu systémů, které jsou optimalizovány pro určitou aplikaci, obecně platí, že každá funkce, která neslouží danému účelu, snižuje spolehlivost (více komponent = nižší spolehlivost) a efektivnost (více potřebné paměti atd.) a je tedy dobré ji eliminovat. Složitost současných systémů však s sebou nese potřebu posunu z nižších do vyšších úrovní abstrakce – od logických členů, přes kombinační a sekvenční moduly (MUX, ALU, registry) až k IPC, procesorům a výpočetním platformám. Zvyšování složitosti jednotlivých stavebních prvků pak znamená, že ne vždy se je všechny podaří plně využít. Např. vyhledávací tabulka s 5 vstupy, nebude plně využita tam, kde je pro implementaci dané logické funkce třeba jen 4 vstupů. Podobně, MPU nemusí pracovat na plném výkonu a využít veškerou dostupnou paměť. Avšak s využitím regulárních stavebních prvků klesá jejich cena. I když se tedy nevyužije jistá část komponent, může být výhodné ji použít.

S použitím složitějších komponent roste úroveň abstrakce návrhu, klesá však efektivita výsledného systému. S omezenou množinou standardních stavebních prvků klesá též diverzita návrhu – existuje menší počet možných řešení. Realizace nějaké složitější funkce pomocí logických členů bude vždy efektivnější s ohledem na kvalitu návrhu (výkonnost, příkon, rozměry), avšak bude pravděpodobně výrazně dražší než např. využití MPU (výjimkou je masová produkce ASIC). Vyšší úroveň abstrakce a používání ověřených komponent je tedy v souladu s trendy na trhu, kde je rychlost uvedení na trh dominantním parametrem ceny. Návrh složitých systémů s využitím komponent a platforem, s důrazem na škálovatelnost je klíčem k efektivnímu využití možností dnešních technologií výroby integrovaných obvodů.

V této kapitole si uveďme nejdůležitější charakteristiky a vlastnosti programovatelného hardware, který tvoří základní integrující prvek výpočetních platforem jak pro tvorbu vestavěných, tak komplexních výpočetních systémů.

3.1 PROGRAMOVATELNÉ LOGICKÉ OBVODY

V roce 1969 byl zveřejněn vynález programovatelných logických obvodů (angl. Programmable Logic Devices – PLD) [62]. Možnost programování HW znamenala konceptuální průlom do způsobu myšlení při návrhu HW. V té době však ještě technologické možnosti výroby integrovaných obvodů neumožňovaly efektivní implementaci PLD. S pokrokem ve výrobě IO se však staly PLD cenově dostupnými a firmy Altera (1983) a Xilinx (1984) je začaly jako první aplikovat v praxi.

Architektury PLD lze rozdělit do dvou základních skupin podle způsobu jejich konfigurace (programování). Pokud si konfigurační informaci obvod interně pamatuje i po odpojení napájecího napětí, hovoříme o nevolatilních PLD. Konfigurační informace je buď definována jednou pro vždy (např. destrukcí propojek) nebo ji lze omezeně měnit (např. paměť konfigurace je typu FLASH). Nevolatilní PLD se typicky používají tam, kde je třeba, aby zahájily svoji činnost bezprostředně po připojení k napájecímu napětí a jako náhrada podpůrné (angl. glue) logiky. Mezi nevolatilní PLD patří např. obvody PAL (angl. Programmable Array Logic), které jsou navrženy pro efektivní implementaci disjunktní formy logických výrazů (suma součinů). Skládají se z programovatelného AND pole (volba mintermu normální disjunktní formy se provádí pomocí destrukce propojek) a pevného OR pole (logický součet vybraných mintermů), které jsou často doplněny o klopné

obvody. Jsou vhodné pro implementaci jednoduchých kombinačních a sekvenčních logických sítí. Obvody GAL (angl. Generic Array Logic) mají obdobnou architekturu, jako obvody PAL, propojky jsou však elektricky programovatelné (řízené pomocí paměťových buněk např. typu EEPROM). A konečně obvody typu CPLD (Complex Programmable Logic Devices) se skládají z AND-OR polí a registrů s elektricky (typicky pomocí paměťových buněk typu FLASH) programovatelnými křížovými přepínači.

Vzhledem k výhodným vlastnostem se dalším textu se soustředíme především na tzv. volatilní PLD typu FPGA, u kterých je konfigurace uložena v paměťových buňkách SRAM a je ji tedy třeba, po každém připojení napájecího napětí, kopírovat z externí paměti. FPGA se skládají z pole konfigurovatelných logických elementů různé složitosti (vyhledávací tabulky, registry, násobičky, malé paměti, logická jádra, procesory, různá rozhraní atd.), programovatelné propojovací sítě a konfigurační paměti.

Jak funkce logických elementů, tak topologie propojovací sítě jsou definovány pomocí distribuované konfigurační paměti. Tato paměť je relativně velká a zabírá poměrně značnou část plochy čipu FPGA. Podobně, i propojovací síť je dimenzována s ohledem na co možno největší stupeň propojitelnosti jednotlivých elementů a zabírá velkou část obvodu. Propojovací síť je též hlavním zdrojem zpoždění a odebírá, díky velkým parazitním kapacitám, podstatnou část dynamického příkonu FPGA (více viz kapitola 6.1). Pro tvorbu logických obvodů tedy v FPGA zbývá poměrně málo plochy čipu, což autor ve své disertační práci [20] ilustroval následujícím vztahem:

$$P_n \approx 10 \cdot P_c \approx 100 \cdot P_l \text{, kde} \qquad (1.)$$

P_n, je plocha, kterou zabírá propojovací síť. P_c je plocha, kterou využívá konfigurační paměť a P_l je plocha využitá pro implementaci logických funkcí.

V reálných aplikacích též nelze plně použít veškeré logické elementy (díky omezeném možnostem jejich propojení) a na základě zkušeností lze konstatovat, že využitelnost se typicky pohybuje mezi 60 – 80 %. Přesto však, díky platnosti Moorova zákona, je využitelná část kapacity FPGA obvodů dostatečná pro tvorbu komplexních výpočetních a paměťových struktur. Technologie FPGA se dnes využívá pro testování procesů výroby integrovaných obvodů a jsou typicky prvními obvody, které se při zavádění nových technologií výroby číslicových IO vyrábí. Příkladem je sdružení firem IBM a Xilinx při zavedení (jako první na světě) 90nm technologie výrobu IO na 300mm křemíkových plátech (wafer) a s měděnými spoji [31]. Při použití FPGA lze testovat všechny parametry výrobního procesu a HW struktur jako je rychlost logických obvodů, dobu přístupu u pamětí SRAM, vlastnosti propojovací sítě, zpoždění spojů atd. FPGA lze též využít pro lokalizaci chybných oblastí na křemíkové podložce pro sledování kvality výrobního procesu. V této oblasti vytlačily FPGA obvody paměti DRAM, u kterých je testování obtížné. FPGA jsou tedy typicky vyrobeny nejpokročilejší dostupnou technologií, ještě dříve než MPU, ASIC a paměti.

Obvody FPGA tedy nabízí nejnovější technologie, což se ve spojení s neomezenou programovatelností, příznivě projevuje v možnostech jejich použití pro tvorbu výkonných a přitom energeticky a cenově dostupných výpočetních systémů. Hlavní výhody jsou shrnuty v následujících odstavcích.

3.2 HETEROGENNÍ VÝPOČETNÍ STRUKTURY

Vzhledem k velké složitosti dnešních aplikací se ukazuje jako nezbytné využit při návrhu systémů s FPGA předem navržených a otestovaných logických obvodů – IPC, které jsou specializované pro jistou funkci. IPC obvody dělíme je dle stupně jejich implementace: Hard IPC (MPU, ALU, násobičky atd.) jsou obvody, které jsou předem rozmístěné a propojené (angl. Placed

& Routed – P&R), což vede na optimální výkonnost a využití plochy čipu. Propojení se zbytkem obvodu je však komplikovanější neboť mají předdefinované vývody. Firemní IPC jsou distribuovány jako seznam komponent a spojů cílové technologie, případně jako strukturní popis v jazyce pro popis hardware (angl. Hardware Descrition Language). Obvod je výsledkem syntézy a mapování na cílovou technologii, ale bez R&P informace. Obvod je možno modifikovat a simulovat a před implementací je třeba provést jejich rozmístění a propojení. U těchto jader lze měnit rozmístění vývodů a přizpůsobit jej tak potřebám zbytku obvodu, ve kterém jsou použity. Výhodou je, že je možno do značné míry jejich výkonnost předem odhadnout a zároveň snadno propojit se zbytkem obvodu. Soft IPC je popisem obvodu v HDL jazyce na úrovni meziregistrových přenosů (angl. Register Transfer Level – RTL). Uživatel tedy musí provést jeho syntézu (součástí distribuce je často skript pro řízení syntézy s ohledem na omezující podmínky) a zajistit jeho P&R a verifikaci.

Při návrhu systémů s FPGA je třeba rozhodnout mezi tím, zda se použije hard IPC, které je výkonné, jasně definované a otestované, či firemní IPC, které je optimálně mapované na cílovou technologii a využívá tedy její nativní struktury (DSP jádra, násobičky, ap.). Případně soft IP jádro, které je potenciálně nejméně výkonné, lze jej však upravit dle potřeby. Každá změna však může též vést na složitější nasazení (validace, P&R).

Pro uživatele je výhodou použití hard IPC zkrácení návrhu obvodu, který danou funkci využívá. Nevýhodou je skutečnost, že volba funkcí, které budou integrovány jako hard IPC a nebudou pro uživatele z nějakého důvodu zajímavé, může způsobit navýšení ceny obvodu, kdy uživatel zbytečně platí za nevyužité funkce. Výrobce FPGA si tedy nemůže dovolit integrovat příliš mnoho různých hard IPC, které nevyužije většina uživatelů. Vzhledem k obrovské škále různých aplikačních domén se proto výběr hard IPC omezuje se jen na základní funkce (MPU, matematické operace, komunikační rozhraní). S ohledem na velké série výroby FPGA však nemusí jisté procento nevyužitých zdrojů v FPGA představovat, s ohledem na cenu, překážku v jeho použití, naopak, může umožňovat budoucí modifikace produktu, který je již u zákazníka. Další výhodou IPC je jejich snadné použití a skutečnost, že jsou předem ověřena, což též zjednodušuje validaci celého systému. Hard IPC standardních procesorů jsou též dobrou volbou pro návrháře, kteří mají k dispozici praxí ověřený (tzv. legacy) kód.

Výpočetní struktury realizované v FPGA se skládají z pole konfigurovatelných logických struktur a konfigurovatelné propojovací sítě. Jejich zrnitost (složitost použitých elementů) určuje efektivitu mapování úlohy do HW pro danou aplikaci. Výhodnou vlastností FPGA je velká volnost ve volbě zrnitosti výpočetních struktur umožňující optimalizovat využití HW. Pokud je výpočet prováděn v FPGA pomocí vzájemně propojených jednoduchých výpočetních elementů, jako jsou malé vyhledávací tabulky (angl. Look Up Table – LUT), multiplexory a jednotlivé registr sdružené v konfigurovatelných logických blocích (angl. Configurable Logic Block – CLB), hovoříme o výpočetních elementech malé zrnitosti (typicky bitově orientované operace). Obvody s malou zrnitostí lze optimalizovat s ohledem na počet použitých elementů, rostou však nároky na propojovací sít. Příkladem systémů, ve kterých se (především díky stupni integrace tehdejších FPGA) používaly výpočetní elementy s malou granularitou, jsou např. systémy MMX, FLEDGE. Pokud se využívá větší zrnitosti jednotek, je výpočet prováděn vzájemně propojenými složitějšími výpočetními elementy (ALU s vícebitovými registry a multiplexory). Je zde třeba méně propojovací sítě než u IPC menší zrnitosti. Na druhou stranu neumožňují optimalizaci na úrovni několikabitových operací. Výhodou je oproti architekturám s jemnou zrnitostí jednodušší programování, nevýhodou je menší flexibilita. Tento přístup se využívá u platforem DX6 a DX64 (viz kapitola 4).

Jako příklad tvorby komplexních výpočetních struktur pro FPGA lze uvést různé architektury procesorů, které byly implementovány, experimentálně ověřeny a aplikovány. Cílem byl výzkum co nejefektivnějšího využití struktur FPGA pro danou aplikační doménu. Získané zkušenosti byly též využity ve výzkumu, v komerčních aplikacích a při tvorbě diplomových prací a ve výuce.

V poslední době se však ukazuje, že je často lepší využít hard a soft IPC MPU dodávané samotnými výrobci FPGA, než navrhovat vlastní architektury procesoru. Výhodou těchto procesorů je skutečnost, že jsou optimalizovány pro danou architekturu FPGA, existují pro ně návrhové nástroje a jejich použití je tedy velmi efektivní (typickými příklady jsou MPU PicoBlaze, MicroBlaze, PowerPC firmy Xilinx [www.xilinx.com]). Cena MPU implementovaného v FPGA může být výrazně nižší než při použití obdobné standardní součástky.

Mezi další výpočetní architektury, které lze optimalizovat pro implementaci v FPGA obvodech, patří škálovatelná systolická pole procesních elementů. Mezi tyto systémy, které byly vytvořeny pro platformy s FPGA v rámci grantů, kterých je autor řešitelem či spoluředitelem, patří např. akcelerátory pro prohledávání DNA sekvencí [41], [38] či pro urychlování grafických operací [75].

3.3 DISTRIBUOVANÁ HIERARCHIE PAMĚTI

S ohledem na potřeby návrhu složitých výpočetních systémů je třeba využívat pamětí se stále větší kapacitou a zároveň krátkou dobou přístupu. Nezbytností je využití temporální (v čase) a prostorové lokality informace v paměťovém prostoru, které otevírají možnost zavedení hierarchie paměti. Hierarchie paměti umožňuje využít existence různých technologií a organizací pamětí s různou kapacitou a rychlostí. Obecně platí, že čím je paměť rychlejší, tím má menší kapacitu.

Pro implementaci výpočetních struktur v FPGA lze, pro potřeby zavedení paměťové hierarchie, s výhodou využít jednak registrů v CLB, distribuovaných pamětí SRAM vytvořených z jednotlivých LUT v CLB, tzv. BlockRAM pamětí (konfigurovatelná paměť SRAM, FIFO atd.) a externích dynamických pamětí DRAM. V případě hard IPC MPU (např. procesory PowerPC v FPGA firmy Xilinx) jsou k dispozici též lokální vyrovnávací paměť cache jak dat, tak programu. Nejnovější technologie FPGA tak umožňují zavádět hierarchii paměti optimalizovanou pro implementaci různých architektur procesorů či aplikačně specifických výpočetních struktur. Připojení i jiných paměťových médií není též prakticky nijak omezeno (např. disky s rozhraním SATA) a lze tedy vytvářet systémy s rozsáhlou virtuální pamětí. Hierarchie a především prostorové rozmístění různých paměťových struktur umožňuje výrazně eliminovat vliv pamětí na rychlost výpočtů, což je hlavní omezením v případě tradičních procesorů (tzv. úzké hrdlo von Neumannovy architektury počítače).

Frekvence, na které pracují logické obvody, se zdvojnásobuje každého 1,5 roku, přičemž za stejné období se např. paměti DRAM zrychlují jen o cca 33 % (cca dvakrát za 10 let) [54]. Z důvodů urychlení přístupu do paměti je nezbytné zařadit mezi hlavní paměť systému a výpočetní jednotky rychlé vyrovnávací paměti s organizací cache či scratchpad. Cílem je maximalizovat výpočet nad rychlou lokální pamětí a omezit výpočty nad pomalou hlavní pamětí. Cache paměti se využívají převážně u architektur určených pro všeobecné počítání. Tyto vyrovnávací paměti mohou být organizovány do více úrovní. Paměti cache se plní kopií části obsahu hlavní paměti automatizovaně (specializovaným HW), dle potřeby programu (za běhu). Jejich obsah tedy není předem explicitně definován. Výhodu principu cache paměti je nezávislost na aplikaci, kdy není třeba předem určovat, která část hlavní paměti se má zkopírovat do cache. Využití cache však není, s ohledem na potřeby konkrétní aplikace, optimalizované a dochází tak k případům, kdy potřebná kopie hlavní paměti není v cache (angl. miss) a potřebná data se musí z hlavní paměti zkopírovat do cache, což je pomalé. Pro aplikačně specifické výpočty je však použití pamětí cache, díky složitosti jejich řízení, většinou neefektivní, neboť většinou lze provádění výpočtu a tím i jeho rozložení časové a prostorové rozložení v paměti předem naplánovat.

Vyrovnávací paměti typu scratchpad se, podobně jako v případě cache, plní dle potřeb aplikace. Tato organizace paměti se často využívá ve specializovaných výpočetních architekturách (např. superpočítač Cray2, většina DSP procesorů) a ve vestavěných systémech (např. herní konzole PS2

firmy Sony). O tom, co bude v kterém čase v této vyrovnávací paměti, se však nerozhoduje za běhu programu (jako u cache), ale předem v době návrhu aplikace. Plnění scratchpad paměti se může plánovat tak, aby bylo prováděno souběžně s výpočtem např. pomocí řadiče přímého přístupu do paměti (angl. Direct Memory Access – DMA).

Cache, na rozdíl od scratchpad paměti, obsahuje jen kopii části hlavní paměti a představuj tedy HW navíc, který nenavyšuje její kapacitu. Oproti tomu scratchpad paměť nahrazuje část hlavní paměti a je tedy s ohledem na cenu a příkon efektivnější než cache, což je u specializovaných výpočetních systémů důležitým atributem. V případě FPGA je vhodné použít cache jen v případě hard IPC MPU (např. PowerPC), které mají integrován jak řadič cache, tak příslušnou paměť SRAM. Pro aplikačně specifické architektury vytvořené pomocí struktur FPGA je lepší využít paměť scratchpad.

Dominantními odběrateli energie ve výpočetních systémech jsou, vedle výpočetních jednotek a propojovací sítě a rozvodů hodinových signálů, především paměti. Se zvyšující se kapacitou paměti nejen, že roste doba přístupu, ale zvyšuje se též jejich příkon. I s ohledem na příkon je výhodou platforem s FPGA možnost využití jak distribuovaných pamětí (není třeba data přenášet mezi hlavní pamětí a výpočetními jednotkami), tak hierarchie, paměti která využívá principu prostorové a temporální lokality dat v paměťovém prostoru. To spolu s prostorovým rozdělením úlohy na více procesů a jejich paralelním vykonáváním, může vést jak na zvýšení výkonnosti, tak snížení odběru elektrické energie viz kapitola 6 . Optimalizací využití paměťových struktur v FPGA umožňuje dosáhnout významného urychlení výpočtů [43] .

3.4 POČÍTÁNÍ V ČASE A PROSTORU

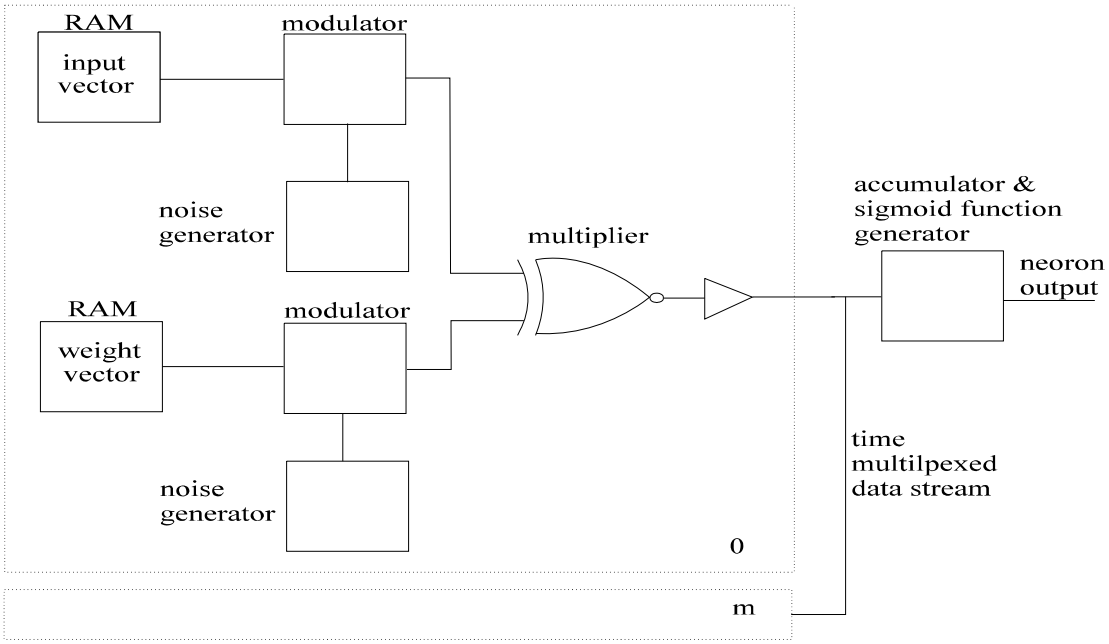
Počítače přinesly průlom do způsobu řešení řady úloh díky možnosti implementovat algoritmy za pomoci malé množiny pevných, předem vyrobených HW struktur, které jsou sdíleny v čase (temporální počítání). Méně komponent vede na vyšší spolehlivost systému a nižší cenu, neboť náklady na jejich pořízení jsou amortizovány jak v čase, tak v různých aplikacích jeho vícenásobným využitím. Řešení úlohy se zjednodušuje na programování SW a ne na tvorbu HW. Sdílení výpočetních prostředků v čase je levné, ale pro řadu aplikací pomalé. Oproti tomu, při čistě prostorovém zpracování se výpočetní prostředky nesdílí, použije se jich tolik, kolik je pro dosažení požadované výkonnosti potřeba, avšak na úkor ceny. Mezi těmito dvěma extrémy leží optimum, kdy je možno dosáhnout zpracovaní úlohy v požadovaném čase (výpočetní výkon) s minimem výpočetních prostředků (cena), při dodržení energetických možností (příkon) a jiných omezení (doba návrhu, atd.). Tato optimalizace je jednou z hlavních náplní HSC návrhu výpočetních systémů, kdy jde o nalezení kompromisu mezi prováděním výpočtu sekvenčně (temporální výpočet) a paralelně (prostorový výpočet).

V případě výpočetních platforem s FPGA lze kombinovat uvedené přístupy velmi efektivně, neboť lze vybrat a naprogramovat právě ty výpočetní prostředky, spolu s distribuovanou a hierarchicky uspořádanou pamětí, které jsou pro danou úlohu nejvhodnější a přitom se nezabývat jejich výrobou.

Vývoj software zabírá dominantní část doby vývoje produktu [26] a představuje tedy i největší položku z ceny návrhu produktu. Pro snížení ceny výrobku se návrháři tradičně snaží minimalizovat cenu použitého hardware. Tento přístup ale má konsekvence ve složitějším návrhu a testování systému. Studie ukazují, že pokud je implementace omezena výpočetními zdroji (paměť, počet hradel), pak výrazně narůstá složitost návrhu a jeho testování [12]. Platí tady, že pokud je stupeň využití výpočetních zdrojů velký, je cena vývoje dominantní položkou ceny celkového vývoje. Pokud ale přidáme další hardware, může výrazně snížit cenu tohoto vývoje. Tato skutečnost nahrává použití platforem s FPGA, kde počet tranzistorů (plocha čipu) není kritickou položkou ceny a zkrácením doby návrhu ušetříme víc, než optimalizací počtu

výpočetních zdrojů. Navíc použití programovatelného HW umožňuje snadné opravy chyb a update i upgrade systému, který je již u zákazníka.

Příkladem optimalizace rozložení výpočtu v čase a prostoru s maximálním využitím distribuovaných pamětí v FPGA je implementace neuronové sítě v FPGA [15] a [19]. Obvod využívá distribuovaných pamětí a výpočet je rozdělen na velké množství sekvenčních výpočtů prováděných paralelně. Využívá se zde tzv. stochastického počítání, kdy jsou čísla representována pomocí modulovaných pseudonáhodných posloupností, ve kterých je pravděpodobnost výskytu log. 1 a log. 0 úměrná jejich hodnotě. Implementace neuronu je pak velmi úsporná, neboť pro operaci násobení stačí použít pouze jeden logický člen XOR a pro akumulaci stačí čítač (blokové schéma neuronu viz Obr. 4).



Obr. 4 Implementace neuronové sítě v FPGA

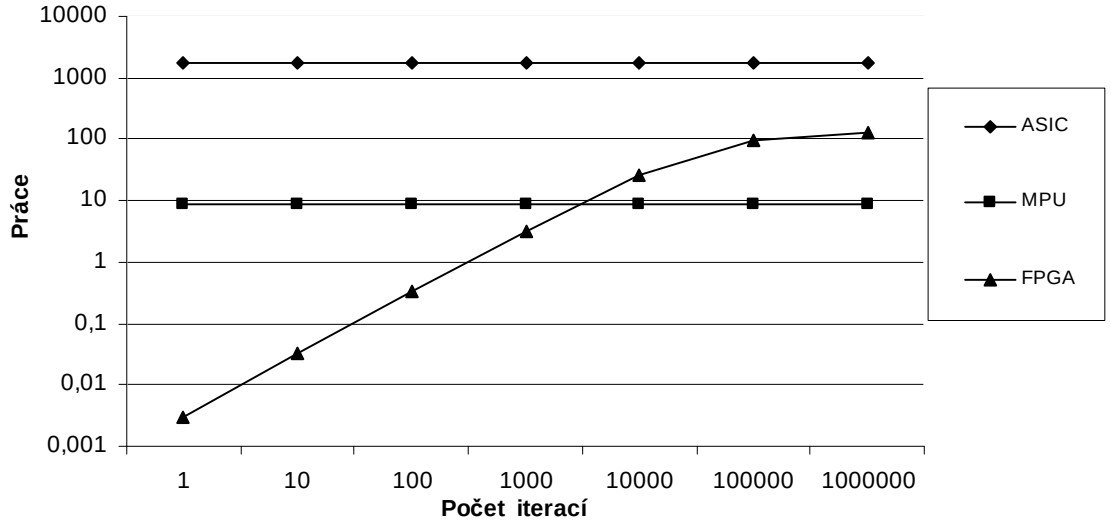
Pro generování pseudonáhodných posloupností, modulátorů a akumulátorů byly v maximální míře využity paměti RAM vytvořené z LUT jednotlivých CLB v FPGA. Díky tomu lze do FPGA implementovat velké množství (stovky) vzájemně propojených neuronů mezi kterými probíhá sériová, časově multiplexovaná komunikace. Výpočet probíhá paralelně ve všech neuronech současně a je proveden za dobu, která je dána požadovanou přesností výpočtu (např. pro reprezentaci 8bitových čísel je třeba využívat náhodné posloupnosti délky 2^{14} bitů).

3.5 REKONFIGURACE

Technologie FPGA otevřela nové možnosti realizace výpočetních systémů díky své konfigurovatelnosti, která umožňuje návrh aplikačně specifických výpočetních architektur (podobně jako u obvodů ASIC) bez nutnosti jejich výroby (podobně jako u MPU). V FPGA jsou konfigurovatelné výpočetní struktury, jejich propojení i řadič optimalizované pro danou úlohu. Pokud provedeme změnu obsahu konfigurační paměti FPGA v době mezi připojením a odpojením napájecího, hovoříme o tzv. rekonfiguraci. Např. FPGA obvody firmy Xilinx [67] podporují rychlost konfigurace až 800 $Mb \cdot s^{-1}$, což vzhledem ke kapacitě konfigurační paměti (až 60 Mb) znamená, že doba konfigurace je dána velikostí obvodu FPGA a pohybuje se v řádech jednotek ms až stovek ms. V řadě případů může být výhodné provádět jen tzv. parciální rekonfiguraci, kdy se konfiguruje jen část FPGA v době, kdy zbytek obvodu pracuje.

Rekonfigurace může být např. použita pro diagnostické účely, kdy se po připojení napájecího napětí zavede konfigurace, která otestuje jak FPGA, tak i okolní obvody a následně se zavede konfigurace řešící danou úlohu (používané např. v inteligentní kameře viz odstavec 4.8). Podobně např. v případě zjištění nějaké poruchy za běhu aplikace se může zavést náhradní konfigurace, která umožní přechod systému na definovanou činnost. Další možností je přepínání mezi konfiguracemi podle stavů, ve kterých se může zařízení nacházet (např. režim nastavení paramentů, režim činnosti, režim úspory energie apod.). Pokud je změna konfigurace prováděna v závislosti na probíhajícím výpočtu dané úlohy, hovoříme o tzv. dynamické rekonfiguraci či rekonfigurovatelném počítání.

Dobu rekonfigurace výpočetních struktur implementovaných v FPGA (výpočetní kontext) je třeba posuzovat v poměru k užitečné práci (době výpočtu). V autorově disertační práci [20] byly odvozeny vztahy mezi efektivitou výpočtů v optimalizovaném HW (ASIC), MPU a dynamicky rekonfigurovaném FPGA. Pro ilustraci si uveďme modelový případ využití tří základních technologií (MPU, ASIC a FPGA) pro výpočet obrazového filtru. Užitečná práce, kterou vykonají příslušné obvody při výpočtu dané úlohy, je vyjádřena v závislosti na počtu iterací (velikost filtrovaného obrázku). Předpokládáme, že užitečnou práci konají pouze hradla, která se účastní daného výpočtu. Počet hradel je pak vztažen na plochu, kterou zabírá příslušný obvod (tzv. normalizovaná funkční hustota D_n). Pro MPU bylo odvozeno, že $D_n(MPU) \approx 100$, neboť výpočet je prováděn typicky v ALU (zanedbejme případy, kdy i přesun dat lze považovat za výpočet – např. bitové posuny pro násobení dvěma atd.) a zbytek obvodu lze z pohledu výpočtu považovat za nutnou režii. V případě obvodů ASIC je $D_n(ASIC) \approx 10.000$, což je 100× více než u MPU, protože zde není téměř žádná režie (uložení a distribuce dat a instrukcí) a prakticky veškerá hradla obvodu se účastní výpočtu. V případě FPGA je režie dána propojovací sítí a konfigurační pamětí a hodnota $D_n(FPGA) \approx 1.000$. Při dynamické rekonfiguraci obvodu FPGA je třeba do režie započíst i dobu rekonfigurace. Závislost užitečné práce (D_n) vykonané příslušným obvodem v závislosti na velikosti úlohy je v grafu na Obr. 5.

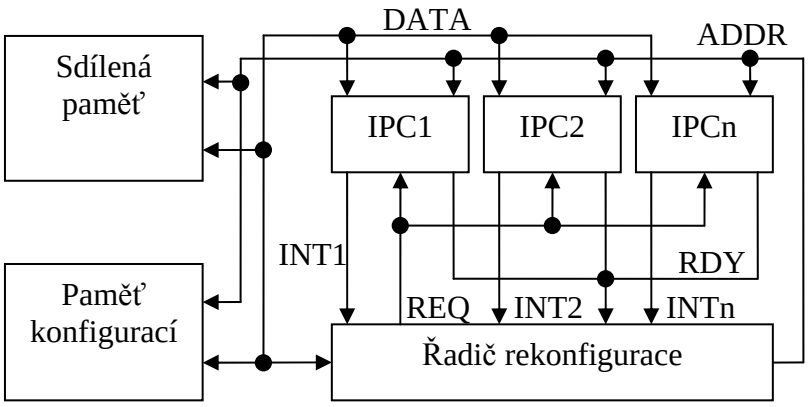


Obr. 5 Závislost užitečné práce na počtu iterací a době rekonfigurace FPGA

Vidíme, že pro danou úlohu FPGA je výkonnější než MPU již při cca 10.000 iteracích a při 100.000 iteracích výpočtu ji lze téměř zanedbat. Výpočet bude proveden cca 10× rychleji než při použití MPU a 10× pomaleji než v případě jeho implementace v ASIC. Výsledkem uvedených analýz je zjištění, že výkonnost FPGA leží mezi aplikačně specifickými výpočetními obvody a plně programovatelnými architekturami (MPU) v poměru cca ASIC:FPGA:MPU=100:10:1.

V grafu na Obr. 5 též vidíme, že výkonnost úloh, které nejsou tvořeny mnoha stejnými operacemi, je omezena rychlostí rekonfigurace FPGA.

Podpoře využití dynamické rekonfigurace byla též věnována široká pozornost v platformách s FPGA, které autor spoluvytvářel. Jednalo se např. o podporu dynamické rekonfigurace platformy DX6 [11], viz řídící obvod na Obr. 6.



Obr. 6 Řadič dynamické rekonfigurace

V daném systému, který využívá spojení CPLD, MPU a FPGA je rekonfigurace řízena z MPU který inicializuje DMA přenosy mezi virtuálně stránkovanou konfigurační pamětí a konfiguračním rozhraním FPGA. Vlastní přenosy provádí řadič implementovaný v obvodu CPLD. Po ukončení konfigurace je vyvoláno přerušení MPU, které signalizuje, že MPU může začít využívat koprocesor či funkční jednotky implementované v FPGA. Je zde podporována jak úplná, tak parciální dynamická rekonfigurace, která může být iniciována i výpočtem probíhajícím v FPGA. Řadič podporuje též komunikaci mezi více výpočetními jednotkami (IPC) v FPGA přes sdílenou paměť. Systém byl např. využit při implementaci systémů pro monitorování dopravy [22]. Vzhledem k charakteru dané úlohy bylo plánování zavádění jednotlivých konfigurací možno provést předem staticky a jejich zavádění realizovat s předstihem pomocí jejich předvýběru. Je podporováno jak zřetězené zpracování výpočtu, kde každému stupni odpovídá jedna z konfigurací, tak zpracování řízené tokem dat.

V řadě případů lze předem naplánovat provádění jednotlivých úloh tak, aby se mohl překrývat výpočet (např. na DSP procesoru) s rekonfigurací FPGA [74]. V následujícím fragmentu programu je znázorněn příklad vykonávání úloh A, B v DSP a X, Y v FPGA:

```
void Process1(const Image *A,Image *OutA)
     {Alloc(UnitX); // prefetch X
     PreprocessInDSP(A);
     Wait(UnitX); // use X now!
     Execute(UnitX,A,OutA);
     Free(UnitX); // free X}
void Process2(const Image * B,Image * OutB)
     {Alloc(UnitY); // prefetch Y
     Wait(UnitY);  // and use immediately
     Execute(UnitY,B,OutB);
     Free(UnitY); // free Y}
```

V Tab. 1 je uveden průběh výpočtu a rekonfigurace, kdy se v časovém slotu 2 provádí rekonfigurace FPGA paralelně s výpočtem úlohy A vykonávané na DSP.

Tab. 1 Průběh výpočetu s využitím dynamické rekonfigurace

Výpočetní jednotka	**Časový slot**						
	1	2	3	4	5	6	7
DSP	A,B	A	A	Idle	Idle	A,B	Idle
FPGA	Idle	Load X,Y	Exec Y	Exec X,Y	Free X,Y	Idle	Idle

Rekonfigurovatelné počítání je motivováno úsporou výpočetních zdrojů jeho temporálním sdílením. Podle způsobu, jakým je rekonfigurovatelný výpočet řízen, lze hovořit o centrálním řízení (angl. controlflow) či řízení tokem dat (angl. dataflow). Centrální řízení je založeno na sekvenčním vykonávání konfigurací, které jsou uloženy v paměti konfigurací a zaváděny do FPGA na základě programu. Výsledky výpočtu jedné konfigurace se předávají do další konfigurace přes vhodné datové úložiště. Konfigurace umožňují realizovat výpočty, které jsou optimálně přizpůsobeny dané aplikaci (od bitově orientovaných až po složité funkční jednotky).

Výhodou rekonfigurovatelného počítání je možnost využití prostorově dekomponované paměťové hierarchie, což umožňuje maximalizovat využití temporální a prostorové lokality dat a představuje největší přínos ve srovnání s tradiční von Neumannovou architekturou počítače. Dynamická rekonfigurace nabízí efektivní implementaci výpočetních struktur, avšak za cenu velmi složitého (a tedy i drahého) návrhu. Obtížné programování rekonfigurovatelných architektur představuje největší překážku v jejich širším použití v komerčních aplikacích. Tento přístup má smysl použít pouze u úloh, u kterých lze předem snadno alokovat výpočetní prostředky a mapovat na ně úlohy a kde se, s ohledem na cenu komponent, vyplatí investovat čas do jejich návrhu.

Na základě zkušeností lze konstatovat, že dynamická rekonfigurace současných FPGA obvodů prozatím nenabízí takový potenciál, aby ji bylo mělo smysl široce komerčně nasadit. Rekonfigurace dnešních FPGA poměrně pomalá, což představuje často neúměrnou režii. V případě použití dynamické rekonfigurace s ohledem na snížení příkonu je třeba uvážit, že proces konfigurace je srovnatelně energeticky náročný jako výběr instrukcí u procesorů (přenos dat z paměti po sběrnici). Autor se uvedenou problematikou dlouhodobě zabýval v době, kdy využitelná kapacita obvodů FPGA byla velmi omezená. S ohledem na stupeň integrace současných FPGA se však ukazuje, že může být efektivnější používat FPGA s větší kapacitou a výkonnost a příkon optimalizovat škálovatelnými výpočetními architekturami viz kapitola 6 . V souladu s tímto názorem je i skutečnost, že komerčně dostupné obvody FPGA spíše nabízí možnost rekonfigurace výpočetních jednotek (DSP jádra, rozhraní) apod. než rychlou rekonfiguraci celé struktury.

Způsobem, jak urychlit proces rekonfigurace FPGA je např. použití více konfiguračních pamětí a zajistit jejich rychlé přepínání, což však vyžaduje integraci dalších konfiguračních pamětí a příslušné propojovací sítě na čipu a není prozatím běžně komerčně dostupné. Vzhledem k tomu, že dominantní část plochy FPGA zabírá propojovací síť, je zřejmé, že zde je největší tlak na inovace. Možným řešením může být použít více propojovacích vrstev a prostorové (3dimenzionální – 3D) integrace. Lze předpokládat, že FPGA budou i v této oblasti průkopníkem, neboť již dnes je možnost 3D výroby integrovaných obvodů reálná viz [32]. Otevírají se tím též možnosti efektivního využití velmi rychlé dynamické rekonfigurace prováděné za běhu aplikace.

3.6 PROVÁDĚNÍ VÝPOČETU

Pro ilustraci možností, které FPGA nabízí, uveďme srovnání způsobu implementace úloh v různých technologiích, viz Tab. 2.

Tab. 2 Způsoby implementace algoritmů

Provádění výpočtu	Řízení	Výpočetní struktury	Komunikace	Realizace
Pevné výpočetní struktury řízené pevným řadičem	Pevné	Pevné	Pevné	Koprocesor k MPU, ASIC FPGA (hard IPC)
Pevné výpočetní struktury interpretující instrukce řízené programem	Program	Pevné	Pevné	MPU FPGA (hard IPC MPU)
Konfigurovatelné výpočetní struktury řízené konfigurovatelným řadičem	Konfig.	Konfig.	Konfig.	FPGA (soft IPC MPU, datová cesta + řadič)
Konfigurovatelné výpočetní struktury řízené konfigurovatelným řadičem, který je řízen programem	Program	Konfig.	Konfig.	FPGA (hard IPC, hard a soft IPC MPU, datová cesta + řadič)

Vidíme, že FPGA umožňují v nějaké formě využití potenciálu různých přístupů k výpočetním architekturám a lze je tedy považovat za integrující technologii hybridních výpočetních platforem. Uvedené členění na ASIC, MPU a FPGA je zde uvedeno s ohledem na porovnání charakteristických vlastností příslušné technologie. Pro úplnost uveďme, že v dnešní době existuje řada technologií, které mají vlastnosti na pomezí výše uvedených architektur. Např. integrované obvody ASSP (angl. Application Specific Standard Produkt) implementují aplikačně specifickou funkci a jsou určeny pro široké spektrum zákazníků. Zde jsou zahrnuty pod ASIC, neboť mají stejný výrobní proces a technologické možnosti, liší se jen cílovou skupinou zákazníků (ASIC je typicky určen pro jednoho zákazníka). Strukturované ASIC obvody (tzv. platformy ASIC) mají velkou část struktury předefinovánu a validovánu a uživatel navrhuje jen jejich konečné propojení a platí tedy jen část nákladů na pořízení masky. V případě MPU neuvažujeme např. architektury s aplikačně specifickým instrukčním souborem (angl. Application Specific Instruction set Processor – ASIP) atd.

3.7 SHRNUTÍ

Prozatím nelze předpokládat, že by dynamicky rekonfigurovatelné architektury mohly aktivně figurovat v systémech pro všeobecné výpočty (osobní počítače). Jednak je jejich využití širokým spektrem uživatelů ovlivněno setrvačností ve využívání již vytvořeného a praxí ověřeného (legacy) SW pro standardní instrukční sady. Podobně jako v případě aplikací, které jsou kompilovány z různých jazyků a spouštěny v operačních systémech, kdy je neefektivní do MPU zavádět různé mikroinstrukce, bude asi nereálné nahrávat do nějakého rekonfigurovatelného stroje různé konfigurace. Lze zde vidět analogii s dynamickým mikroprogramováním u historických MPU, jehož koncepce je založena na myšlence, že pro každý použitý vyšší programovací jazyk a případně i řešenou úlohu, je vhodné použít specializovaný soubor mikroinstrukcí. Tento aplikačně specifický instrukční soubor se zavede do MPU, které tedy bude vykonávat pouze instrukce, jež jsou ve zpracovávané úloze použity, což může vést na vyšší efektivitu využití výpočetních struktur. Tento koncept však nebyl úspěšný, neboť je s ním spojena velká režie při přepínání kontextu (všechny programy nejsou napsány stejným jazykem a kompilovány stejným kompilátorem a je tedy třeba zavádět pro každou úlohu nové instrukce). Podobně bude asi neefektivní na nějakém rekonfigurovatelném stroji přepínat rychle mezi více úlohami kdy

zavádění konfigurace ad hoc znamená neúměrnou režii s přepínáním kontextu. Rekonfigurovatelné počítání bude tedy stále spíše atraktivní pro aplikačně specifické úlohy.

Dalším problémem bránícím širokému nasazení rekonfigurovatelného počítání je skutečnost, že pro co nejlepší využití potenciálu, který návrh aplikačně specializovaného HW nabízí, je pro jejich specifikaci prozatím dominantní použití HDL jazyků. Efektivní použití těchto jazyků vyžaduje poměrně hluboké znalosti z oblasti návrhu HW, které nelze vyžadovat u všech programátorů v oblasti SW. Z tohoto důvodu se dnes objevují snahy o použití tradičních SW jazyků jako je C pro popis HW/SW systémů (viz kapitola 5). Jejich použití sice může zvýšit produktivitu práce návrháře, ale obecně vede na méně efektivní systém.

Výrobní technologie integrovaných obvodů se neustále vylepšují a současné FPGA využívají stále většího počtu tranzistorů, což do značné míry omezuje vliv režie způsobené potřebnou propojovací sítí a konfigurační pamětí. Díky tomu lze předpokládat, že platformy na bázi FPGA obsahující hard a soft IPC, spolu s programovatelnou propojovací sítí, budou postupně dominovat návrhu hardware a HSC techniky se stanou jeho nezbytnou součástí.

S rostoucím počtem zařízení využívajících FPGA technologii bude rychle klesat její cena. Je to dáno tím, že pro různé aplikace bude použita stejná platforma a cena jejího návrhu a výrobních technologií se rozpočítá mezi všechny uživatele (podobně jako v případě MPU). Oproti tomu u obvodů ASIC bude stále platit, že náklady na masky se musí rozpočítat do dostatečného počtu vyrobených kusů, aby se vůbec vyplatila. Podobně jako u obvodů ASIC, je i v případě MPU nutno neustále investovat do vývoje a výroby nových verzí. U MPU je stabilní pouze instrukční sada, ale HW se musí neustále vylepšovat (nižší příkon a větší výkonnost). To znamená, že jejich cena bude relativně vyšší než hard IPC MPU se standardními ISA integrovaných ve velkém počtu v platformách FPGA.

Objevují se též snahy o implementaci programovatelných HW struktur v obvodech ASIC. Díku tomu se pravděpodobně se bude stírat hranice mezi FPGA a tzv. systémy na čipu (angl. System on Chip - SoC) v tom smyslu, že FPGA budou obsahovat stále více hard IP jader, což umožní dosahovat vysokých výkonností podobně jako v případě ASIC. Počet aplikací využívajících FPGA již dnes převyšuje nad ASIC aplikacemi. Předpokládá se, že počet návrhů s využitím FPGA poroste z více než 80.000 v roce 2005 na více než 110.000 v roce 2010 i přesto, že ASIC zatím přináší větší zisk [44].

V poslední době se objevují snahy o tvorbu obvodů, které mají část fixní (ASIC) a část na bázi FPGA. Fixní část je určena pro tu část určité třídy aplikací (např. komunikačních zařízení), která se nemění (např. fyzické rozhraní) a FPGA pro potřeby modifikací (např. protokol). I zde bude nejspíš platit, že FPGA budou dominovat, protože i ASIC/FPGA obvody jsou stále spíše vhodné pro velmi omezenou aplikační doménu.

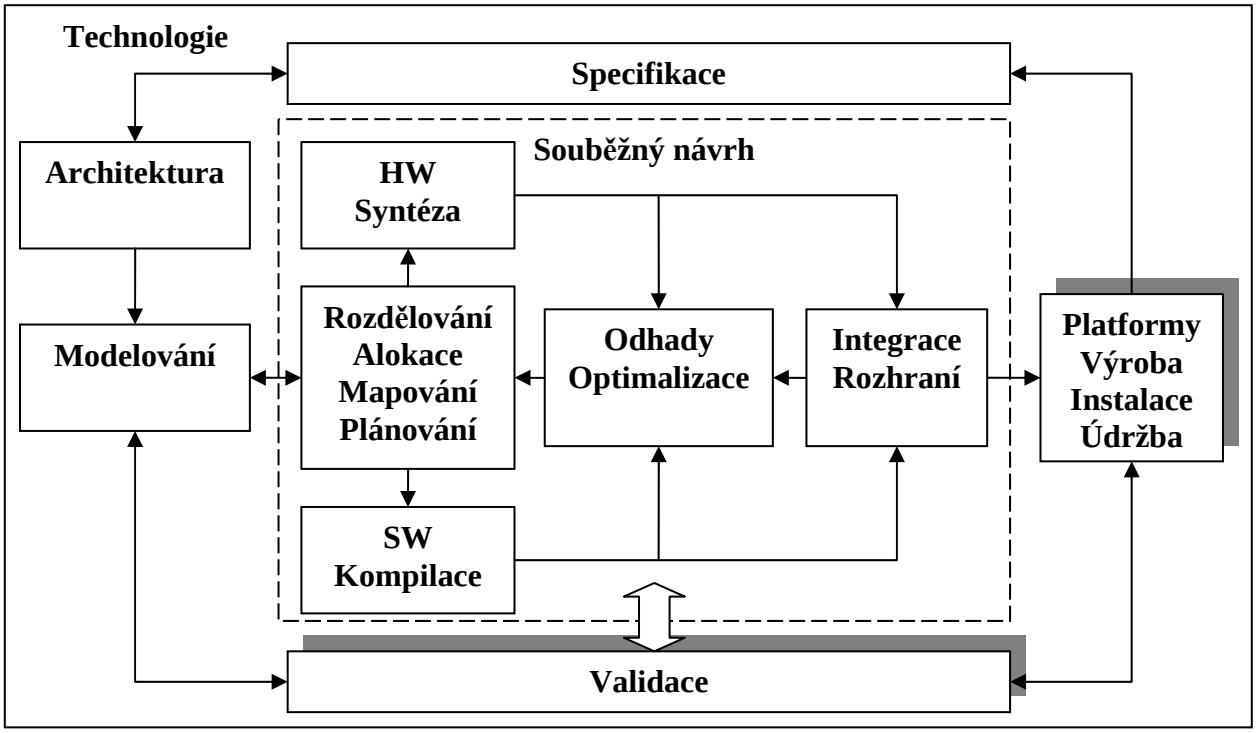
Dalším důvodem hovořícím ve prospěch technologií FPGA je právě cena návrhu, která je díky programovatelnosti (a tedy i rychlá odezvě na potřeby trhu) výrazně nižší u obvodů FPGA než ASIC. Cena komponenty FPGA je dnes méně než 50 dolarů za 1 milion ekvivalentních hradel [56] a nepředstavuje tedy v porovnání s obvody ASIC velkou nevýhodu. V případě FPGA obvodů, je cena návrhu a masky rozpočítána do velkého množství prodaných obvodů. Nižší výkonnost též nemusí být nevýhodou obvodů FPGA oproti obvodům ASIC, neboť právě díky nízké ceně komponent FPGA lze často využít většího počtu hradel pro daný návrh (viz kapitola 6). Problém lze však spatřovat v tom, že FPGA mají (především díky propojovací síti), větší příkon než funkčně ekvivalentní ASIC. V dalších kapitolách budou diskutovány možnosti vylepšování jak výkonnosti, tak příkonu výpočetních systémů implementovaných v obvodech FPGA aplikací vhodného modelování, návrhových technik a výpočetních platforem.

4 PLATFORMY

Pro experimentování s rekonfigurovatelným počítáním, výuku a pro potřeby realizace komerčních aplikací byla vyvinuta řada výpočetních platforem s FPGA, na jejichž koncepci, návrhu, realizaci a použití v praxi se autor podílel, viz následující odstavce.

V poslední době se myšlenka platforem v literatuře široce diskutuje především v kontextu SoC [58]. Myšlenka platforem však není nová, viz systémy MMX a FLEDGE uvedené v kapitole 1 . I osobní počítače jsou jistou platformou, na které je postavena většina dnešních výpočetních systémů pro všeobecné počítání. Posun je však ve stupni integrace, složitosti a heterogenitě, které dnešní výpočetní platformy nabízí.

Použití platforem s FPGA, které lze plně programovat (konfigurovat), otevírá nové možnosti rychlé tvorby efektivních výpočetních systémů. HW systému totiž není třeba vyrábět, ale stačí je jen (spolu se SW) naprogramovat. Jedná se o evoluci konceptu opětovného použití ověřených komponent umožněný vyšší hustotou integrace integrovaných obvodů. Kontext problematiky platforem při HSC návrhu výpočetních systémů je znázorněn na Obr. 7.



Obr. 7 Platformy

Platformy představují jistý posun v tradičním pohledu na HSC, kdy se systém navrhuje shora-dolů (napřed specifikace, pak implementace). Jedná se spíše o postup zdola-nahoru, kdy existuje předem vyrobená platforma, která se programuje dle potřeby. Zkušenosti ukazují, že tento přístup umožňuje jak dosažení vyššího stupně opětovného využití návrhu (snadná tvorba rodin produktů viz kapitoly 12 a 13), tak i jeho vysokou kvalitu. Platformy s FPGA částečně omezují stupeň volnosti v návrhu ale bez úplného omezení flexibility. Je to kompromis mezi přístupem zdola-nahoru a shora-dolů. Konfigurovatelné platformy však přispívají ke zkrácení doby návrhu a tím i ceny.

Koncept platforem, autorem poprvé aplikován v roce 1994, byl postupně, s dostupností nových technologií, rozšiřován o nové architektury. Na základě zkušeností s využitím výpočetních platforem ve velkém množství produktů (desítky aplikací, stovky systémů), lze konstatovat, že s rostoucími požadavky moderních aplikací, budou heterogenní výpočetní platformy, které se skládají z výpočetních jednotek s různou složitostí (MPU, IPC a konfigurovatelných bloků), jsou

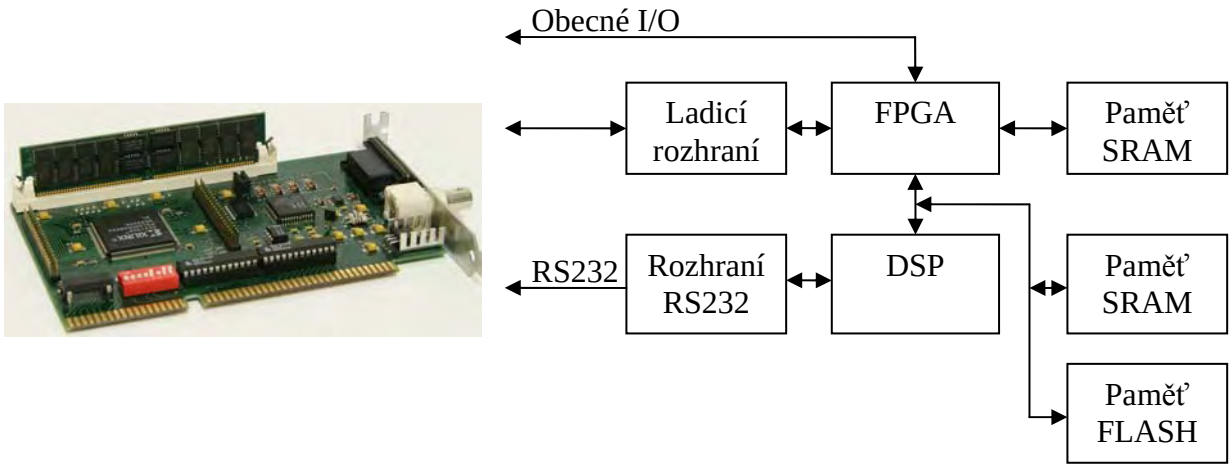
škálovatelné, efektivní (příkon, výkonnost, atd.) a nabízí řadu dalších vlastností (např. snadné integrace s jinými obvody díky podpoře prakticky všech používaných standardů logických úrovní), nabývat na důležitosti.

Platformní HSC návrh byl též aplikován na systémové úrovni, kdy např. vizuální systémy (viz kapitoly 12 a 13) jsou postaveny na platformě tvořené inteligentními kamerami, FPGA výpočetními jednotkami a softwarem pro číslicové zpracování obrazu. Flexibilita těchto platforem umožnila snadnou tvorbu celých rodin systémů.

4.1 GX

Platforma GX je realizována na kartě do PC (ISA sběrnice) a využívá FPGA pro digitalizaci a předzpracování obrazu z kamer, synchronizaci činnosti kamer, výpočetních jednotek s činností výrobních linek a řízení osvětlovacích jednotek. Bylo navrženo několik generací, které byly použity jako integrující prvek vizuálních systémů pro kontrolu výroby na výrobních linkách [73].

První generace[5] těchto desek (1996) umožnily, díky programovatelnosti FPGA technologie, vytvoření flexibilní architektury pro tvorbu rozsáhlých vizuálních systémů. Díky tomu bylo možno postupně přidávat nové funkce podle toho, jak rostly požadavky uživatelů a potřeby nových aplikací. Koncept karet s FPGA jako integrujícího prvku vizuálních systémů se úspěšně využívá dosud (desítky instalací, obrat přes 150 mil. Kč).

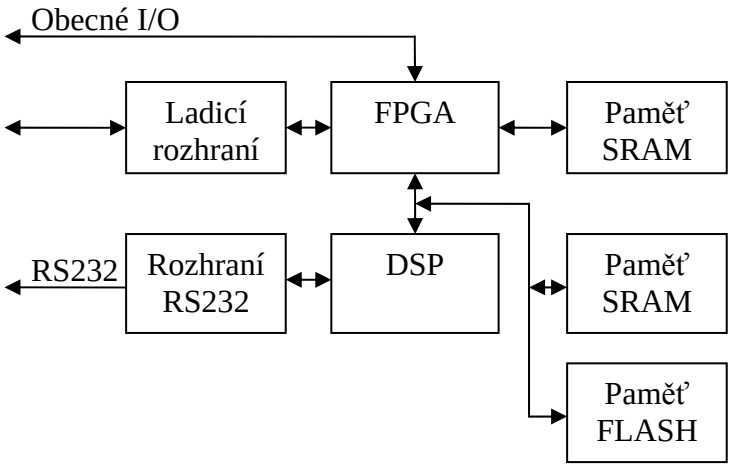


Obr. 8 Platforma GX

4.2 DSPX

Platforma DSPX využívá spojení obvodu FPGA (řady XC4000 firmy Xilinx) a DSP procesoru (TMS320C32 firmy Texas Instruments). FPGA je mapován jako koprocesor do adresového prostoru DSP a transfer dat je podporován až 6 DMA řadiči. Obvod FPGA má též k dispozici lokální paměť scratchpad, kterou využívají výpočetní jednotky implementované v FPGA. Blokové schéma a fotografie systému DSPX je na Obr. 9.

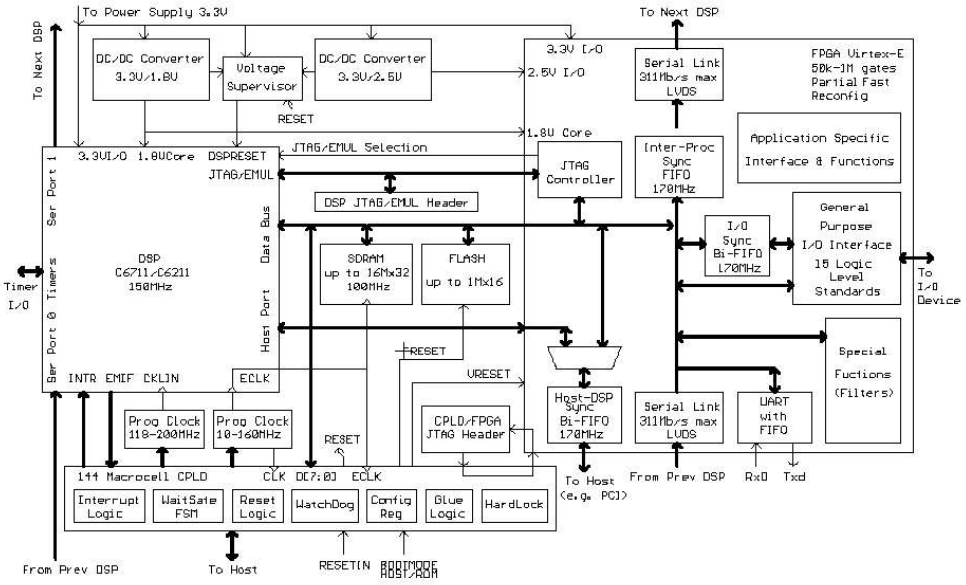
[5] Později byly nahrazeny novou řadou karet pro PCI sběrnicí a FPGA s větší kapacitou, které již autor nerealizoval.



Obr. 9 Platforma DSPX

Architektura platformy je optimalizována pro aplikace z oblasti číslicového zpracování signálů a s její pomocí byla realizována řada inženýrských děl. Mezi nevýznamnější patří systém záznamu a on-line indexace signálů prováděné pomocí číslicových filtrů pro Ministerstvo vnitra ČR. Analýza signálu probíhá v reálném čase na DSP a záznam dat je prováděn na pevný disk, který je řízena z FPGA.

Další aplikací platformy DSPX je systém [21] pro měření vibrací pomocí LVDT senzorů (lineární proměnný diferenciální transformátor). U tohoto projektu se rozdělením úlohy na FPGA a DSP a aplikací algoritmů číslicového zpracování signálu podařilo dosáhnout výrazně lepších parametrů než u komerčně dostupných řešení (tento systém byl vytvořen ve spolupráci s pracovníky Pennsylvania State University, USA, pro společnost Lord Corporation, Inc., USA).



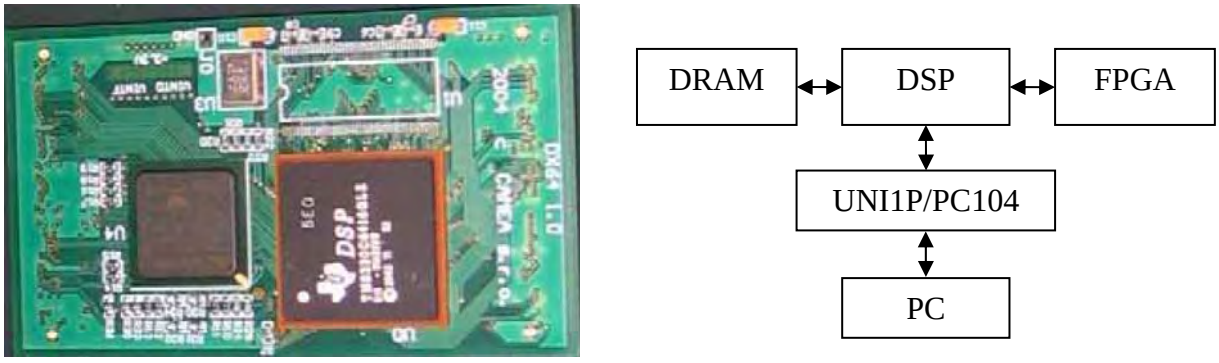
Obr. 10 Platforma DX6

4.3 DX6

Platforma DX6[6] byla navržena pro potřeby vysoce náročných výpočtů při zpracování obrazu v reálném čase. Deska je osazena VLIW DSP procesorem řady C6211 (pevná řádová čárka) nebo C6711 (plovoucí řádová čárka) firmy Texas Instruments, FPGA řady Virtex-E nebo Spartan-E, pamětmi SDRAM a FLASH a obvod typu CPLD. Platforma byla použita ve výzkumných projektech např. [76], a řadě komerčních aplikací viz např. [75], ve kterých bylo použito cca 1500 těchto platforem za desítky mil. Kč. Blokové schéma a fotografie platformy DX6 je na Obr. 10.

4.4 DX64

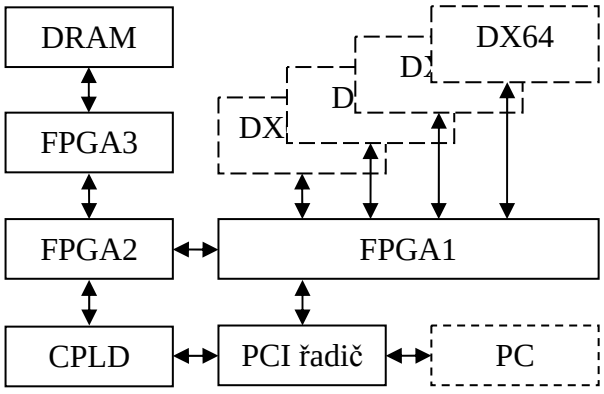
Deska DX64 je koncipována jako nesuvný modul pro vestavění do větších systémů. Na modulu je integrován DSP procesor řady C64 firmy Texas Instruments, FPGA ViretxII firmy Xilinx, paměť DRAM a podpůrné obvody. Modul lze nasunout do desek UNI1P a či PC104 viz dále. Karta byla se použita v řadě komerčních projektů a při výzkumu [74]. Blokové schéma a fotografie karty DX64 je na Obr. 11.



Obr. 11 Platforma DX64

4.5 UNI1P

Do této desky lze vložit až 4 moduly DX64. Dále obsahuje přídavné obvody FPGA pro implementaci rozhraní a IP jader. Obvod CPLD zajišťuje, ve spolupráci s obvodem PLX (rozhraní PCI), transfer dat mezi PC a UNI1P a pamětí DRAM.



Obr. 12 Blokové schéma platformy UNI1P

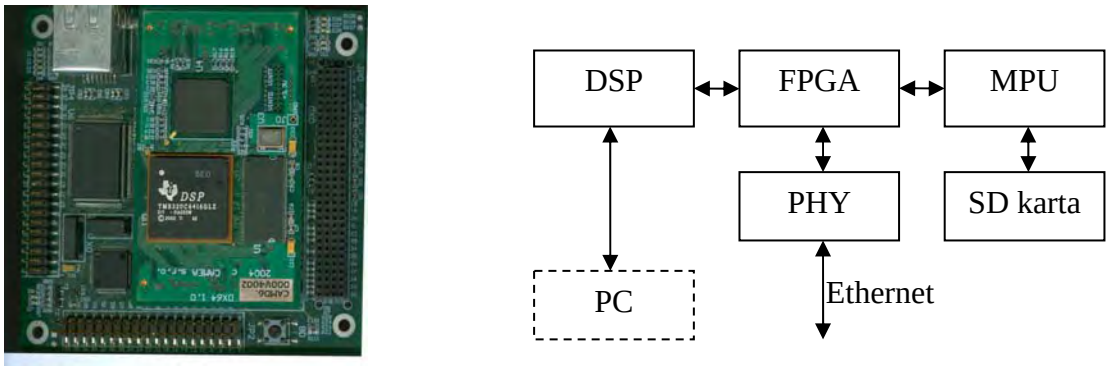
[6] Firmaware platformy bylo vyvíjeno ve spolupráci s firmou Tescan, spol. s r.o., která je používá ve svých mikroskopech.

Deska může být vložena do PC a kooperovat s aplikačním softwarem či pracovat autonomně. Tento velmi výkonný systém (4 DSP procesory, 10 FPGA obvodů a PC) umožňuje tvorbu akcelerátorů v řadě oblastí. Byl nasazen jak v komerčních aplikacích, tak použit při výzkumu [74]. Blokové schéma platformy UNI1P je na Obr. 12 a fotografie na Obr. 13.

Obr. 13 Fotografie platformy UNI1P

4.6 PC104

Tato platforma je tvořena spojením desky DX64 s rozhraním Ethernet 1Gb·s^{-1}, přídavnými pamětmi a mikrokontrolérem. Může být spojena např. s průmyslovým PC se sběrnicí PC104. Blokové schéma a fotografie platformy PC104 je na Obr. 14.



Obr. 14 Platforma PC104

4.7 COMBO6

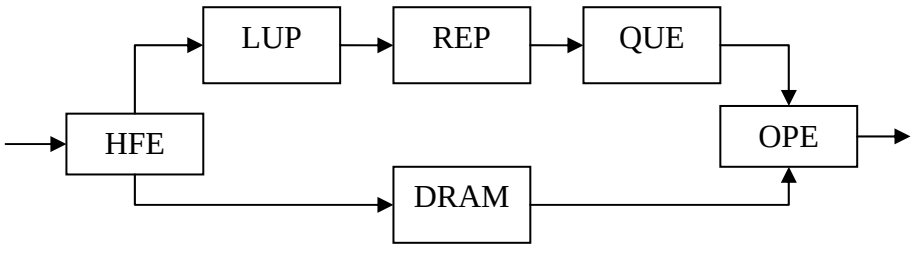
Platforma COMBO6 [46], [47] byla vyvinuta pro potřeby výzkumného projektu Liberouter[7] [www.liberouter.org]. Původním cílem projektu bylo zvýšit propustnost směrovače síťových paketů realizovaného na PC. K tomuto účelu byl navržen a realizován HW akcelerátor, který urychluje časově kritické části směrovače (směrování paketů, firewall). Jedná se o desku, která se v PC instaluje na sběrnici PCI viz Obr. 15. Akcelerace úloh je prováděna v FPGA, ke kterému jsou připojeny pomocné scratchpad paměti SRAM, hlavní paměť DRAM, asociativní paměti CAM a podpůrné obvody. Při návrhu HW platformy COMBO6 byly využity zkušenosti z návrhu a použití platforem DX6 a DX64.

[7] Autor byl jedním z iniciátorů projektu a je spoluautorem jeho původní koncepce a architektury HW. Na aktivitách projektu Liberouter se v současnosti podílí pouze jako externí spolupracovník. Projekt Liberouter je sponzorován sdružením CESNET, s.z.p.o. a řadou EU grantů viz [www.liberouter.org].

Obr. 15 Fotografie platformy COMBO6

Urychlování směrování je prováděno tak, že přepínání paketů a jejich filtrování je prováděno ve výpočetních jednotkách v FPGA. V PC se pak softwarově vykonává zbytek požadovaných činností, jako je kalkulace směrovacích tabulek, konfiguraci FPGA, počítání statistik a ošetření výjimek.

Zpracování paketů v HW je prováděno pomocí řady specializovaných výpočetních jednotek (tzv. nanoprocesorů), jejichž složitost je někde mezi složitostí konečných automatů doplněných o aritmetické a logické jednotky (viz odstavec 5.2) a jednoduchým RISC procesorem. Výhodu tohoto přístupu je možnost měnit jejich chování i za běhu aplikace. Každý paket je analyzován v sadě zřetězených funkčních jednotek. Na vstupu je FIFO paketů (angl. Input Packet Buffer – IPB), následuje extrakce hlavičky paketu (angl. Header Field Extractor – HFE) odkud je tělo paketu ukládáno do paměti DRAM. Podle typu hlavičky je následně rozhodováno, co se má s příslušným paketem provést. Vyhledávací procesor (angl. Lookup Processor – LUP) zpracovává unifikovanou hlavičku pomocí předepsaného programu uloženého v asociativní paměti CAM. Použití CAM pamětí urychluje vyhledávání ve směrovací tabulce. Její kapacita je omezena na 300 bitů, což je pro potřeby implementace síťového protokolu IPv6 málo (je třeba téměř 600 bitů). Řešením tohoto problému bylo provedeno optimalizací vyhledávání s pomocí CAM paměti a podmíněných skoků v jednotce LUP [3]. Hlavičky paketů mohou být v jednotce REP (angl. Packet Replicator) a ve výstupní frontě FIFO (angl. Output Queues – QUE) dále replikovány na více portů. Ve výstupní jednotce OPE (angl. Outrut Packet Editor) je modifikována hlavička paketu před odesláním na výstupní port. Blokové schéma směrovače je na Obr. 16.



Obr. 16 Blokové schéma směrovače s platformou COMBO6

Pokud nelze paket zpracovat v HW desky, může být přesměrován ke zpracování do PC. Tyto výjimky představují pakety, jejichž směrování není definováno na úrovni FPGA. Pokud je takový paket v FPGA detekován, je vyvoláno přerušení a SW na PC si jej převezme k dalšímu zpracování.

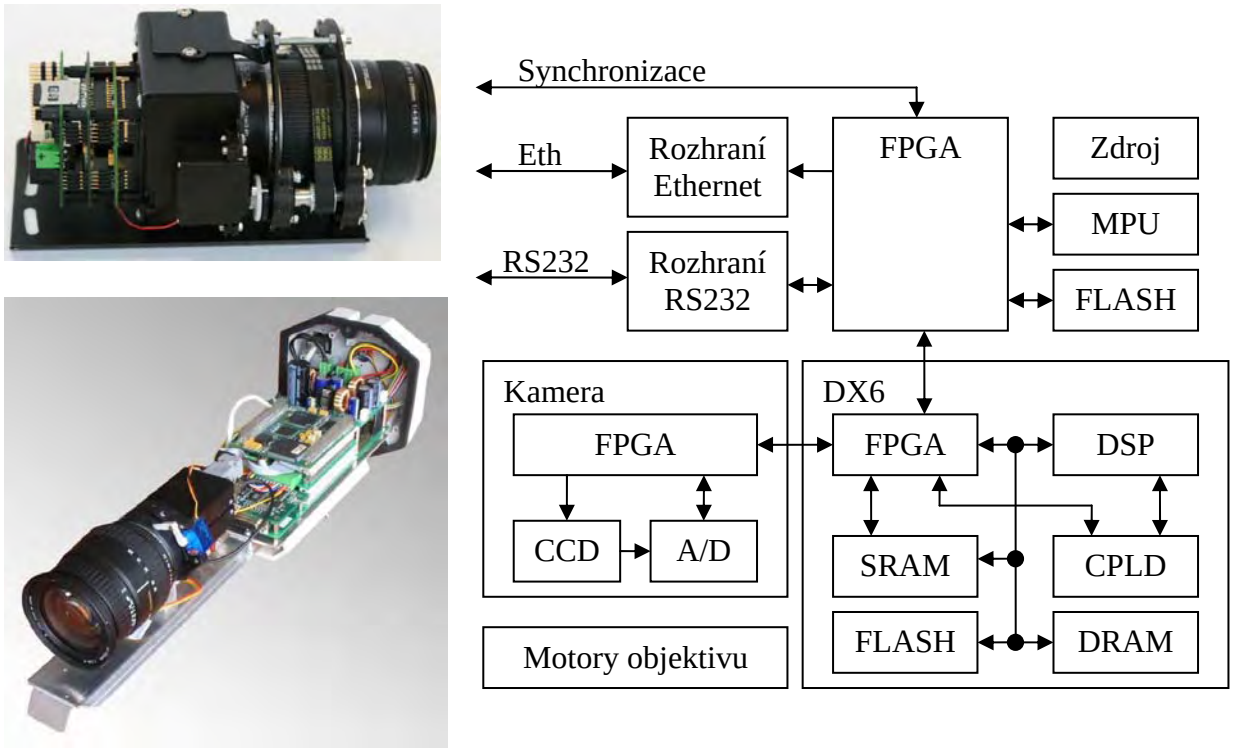
Platforma COMBO6 může být osazena přídavným modulem s různými rozhraními, viz výzkumná správa [50]. Jedná se např. o čtveřici 1 $G \cdot b^{-s}$ portů s metalickým [48] či optickým rozhraním [49]. Součástí systému je též SW nadstavba pro konfiguraci FPGA, přístup do pamětí na desce COMBO6, atd.

Z pohledu HSC technik implementace úloh bylo použito strategie napřed vše v SW s přesuny časově kritických částí úlohy do HW [18]. Jedná se o princip urychlování často vykonávaných částí úlohy viz odstavec 6.4. Směrovač byl nejprve implementován v SW a běžel na PC, následně se profilací identifikovaly časově kritické části směrovacího algoritmu, které se postupně přesunovaly do HW na desce COMBO6. Tím se dosáhlo stavu, že pro běžný provoz na síti lze podstatou část paketů zpracovat v HW a zbytek v PC, čímž se dosáhlo výrazného urychlení směrování. Systém lze též škálovat replikací jednotek pro více portů.

Z pohledu autora asi nejdůležitějším aspektem projektu Liberouter je skutečnost, že se podařilo vychovat několik generací studentů a výzkumníků, kteří mají zkušenosti s metodikou HSC i návrhem komplexních systémů s využitím programovatelného HW na bázi FPGA. Právě díky těmto zkušenostem se objevuje řada nových výzkumných aktivit, které využívají programovatelný hardware (více viz www.liberouter.org).

4.8 INTELIGENTNÍ KAMERA UNICAMD

Při vývoji multifunkčních vizuálních systémů pro monitorování dopravy byly aplikovány HSC techniky pro optimalizaci zpracování algoritmů [75]. Pro předzpracování obrazu v místě kamery byla využita platforma DX6, která byla vestavěna do kamery[8], viz Obr. 17.



Obr. 17 Platforma UnicamD

Kamera má HDTV rozlišení, což představuje 4× větší objem zpracovávaných dat než v případě běžné TV kamery a klade tedy enormní nároky jak na výkonnost výpočetních jednotek, tak na

8 Vývoj byl částečně financován z grantu EUREKA E!3625 - INTELLIVIDEO. MŠMT kód: OE 219. 2006 – 2008.

kapacity přenosových tras. V DX6 je implementován algoritmus, který hledá v obraze příznaky charakteristické pro detekci registrační značky vozidla. Díky tomu není třeba přenášet snímky pořízené kamerou ke zpracování na centrálním serveru, ale je možno je předzpracovat přímo v kameře a pro archivaci a další zpracování předávat jen ty, které nesou informaci relevantní pro danou aplikaci (více viz odstavec 5.3).

V kameře je též integrováno rozhraní Ethernet 100 $Mb \cdot s^{-1}$ a v DSP na desce DX6 je implementován síťový protokol TCP/IP. Pro komunikaci lze tedy využít běžně dostupné komunikační infrastruktury (i bezdrátové sítě operátorů). Spojením HDTV kamery, výpočetní platformy DX6 a síťového rozhraní vznikla nová platforma UnicamD (v současnosti je k dispozici i novější varianta UnicamD2), která je využívána jako základní stavební prvek systémů pro monitorování dopravy. Díky flexibilitě systému UnicamD byla postupně realizována řada různých aplikací, viz kapitoly 12 a 13 . Kompaktní konstrukce přinesla řadu výhod i s hlediska jednoduché instalace a spolehlivosti. Použití FPGA obvodů umožnilo provádět update a upgrade celého firmware kamery na dálku. Dalším klíčovým aspektem návrhu a použití platformy UnicamD je využití GALS technik pro snížení příkonu (viz odstavec 6.4), jejichž aplikace byla usnadněna vhodným modelováním výpočetních úloh (viz odstavec 5.2). Snížení příkonu otevřelo možnost nasazení kamerových systémů i v místech, ve kterých je obtížné či nemožné zajistit trvalé napájecí napětí. Nízký příkon umožnil využít pro napájení baterií, které se nabíjí ze sloupů veřejného osvětlení, které je dostupné prakticky všude. Právě skutečnost, že pro kamerové systémy s platformu UnicamD není třeba speciální infrastruktury (bateriové napájení díky nízkému příkonu a bezdrátový provoz díky předzpracování obrazu), představuje klíčovou výhodu na trhu a fakticky přispělo k vítězství v řadě veřejných soutěží. V současné době jsou v rutinním provozu v terénu nasazeny stovky těchto kamerových platforem.

4.9 FITKIT

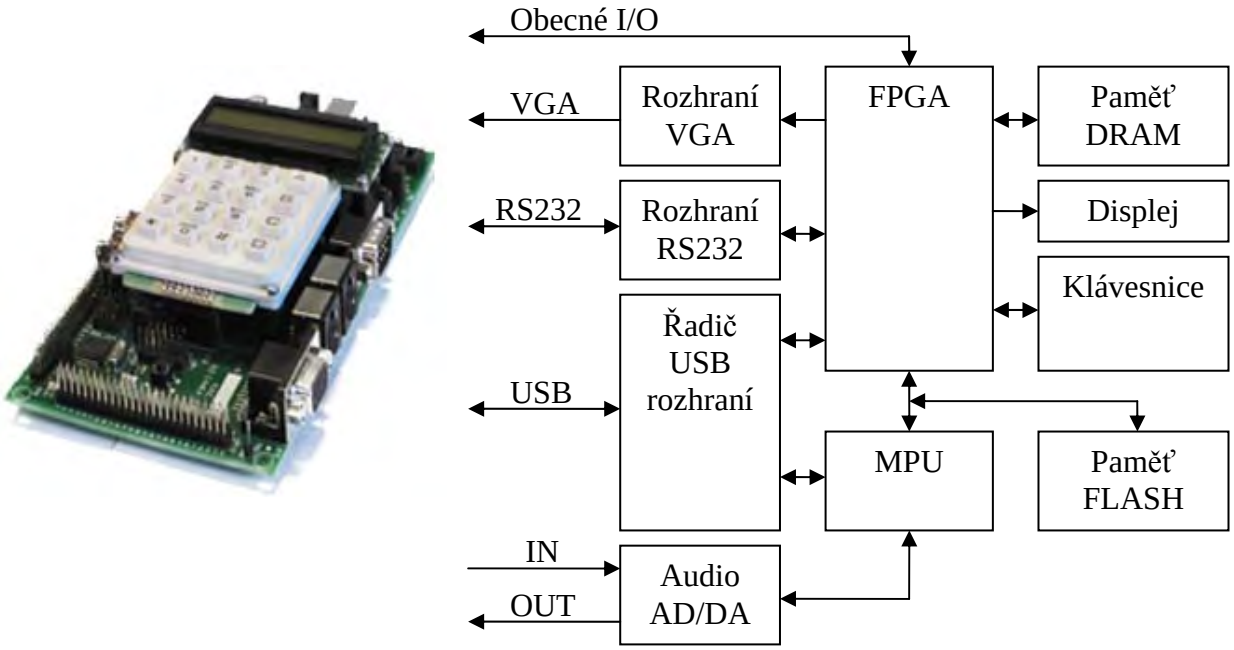
Zkušenosti z výsledků výzkumu a použití výpočetních platforem s FPGA se též podařilo aplikovat ve výuce. Vzhledem k počtům studentů na FIT VUT v Brně je velmi obtížné zabezpečit prezenční laboratorní výuku. Tohoto důvodu byla pro potřeby zkvalitnění a zefektivnění praktické výuky technicky orientovaných kurzů navržena a zavedena do výuky platforma s názvem FITkit[9] [https://www.fit.vutbr.cz/kit]. Hlavní motivací bylo, aby všichni studenti měli k dispozici svůj „osobní“ výukový kit, se kterým budou moci pracovat ve škole, doma i na kolejích. Kit pak studenta provází po celou dobu studia napříč bakalářským i magisterským studijním programem. Znalosti a návyky, získané ve výuce, jsou využívány pro tvorbu ročníkových, semestrálních a diplomových prací a po skončení studia budou použitelné v praxi. Jde o zcela nový přístup ve výuce technicky zaměřených předmětů studijních programů na FIT VUT v Brně.

Výuková platforma FITkit je navřena s ohledem na potřeby HSC návrhu. Obsahuje proto 16 bitový MPU s nízkým příkonem a řadou periferií a programovatelný hardware (FPGA). Software po MPU se tvoří v jazyce C a do spustitelné formy se překládá pomocí GNU překladače. Generování konfiguračních dat pro FPGA z popisu v jazyce VHDL probíhá pomocí volně dostupných profesionálních návrhových systémů. Veškerý potřebný software pro práci s kitem je tedy k dispozici zdarma.

V současnosti je, či v průběhu roku 2008 bude, kit využit ve více než 10 kurzech. Do poloviny roku 2008 bylo vypsáno, přiděleno či již obhájeno více než 70 studentských projektů, které přímo využívají výukový kit. Dále byl plně zprovozněn a je rutinně využíván web s veškerými

[9] Autor je iniciátorem konceptu výukové HSC platformy s FPGA, organizačně zabezpečil její návrh a výrobu a je spoluautorem návrhu HW. Jedná se však o kolektivní dílo (viz neustále se rozrůstající realizační tým www.fit.vutbr.cz/kit/tym.html).

informacemi o kitu. Veškeré výsledky práce studentů s kitem jsou přístupné na internetu ve zdrojové formě pro kohokoli. Byl zaznamenán zájem jak studentů ze středních i jiných vysokých škol než FIT, tak odborníků z praxe o zakoupení kitu pro studijní účely. Díky vstřícnému přístupu fakultní knihovny, byl zaveden systém evidence zápůjček studentům. Celkem bylo mezi studenty distribuováno přes 1.600 ks kitů, které byly z části financovány z prostředků FIT VUT v Brně, dále pak z grantu FRVŠ (FRVŠ A2331/2007 : Kity pro podporu výuky technicky orientovaných kurzů. 2007) a z projektu EU „Zvýšení konkurenceschopnosti IT odborníků absolventů pro Evropský trh práce“, registrační číslo CZ.04.1.03/3.2.15.1/0003, který byl spolufinancován Evropským sociálním fondem (ESF) a státním rozpočtem České republiky. Fotografie a blokové schéma platformy FITkit je na Obr. 18.



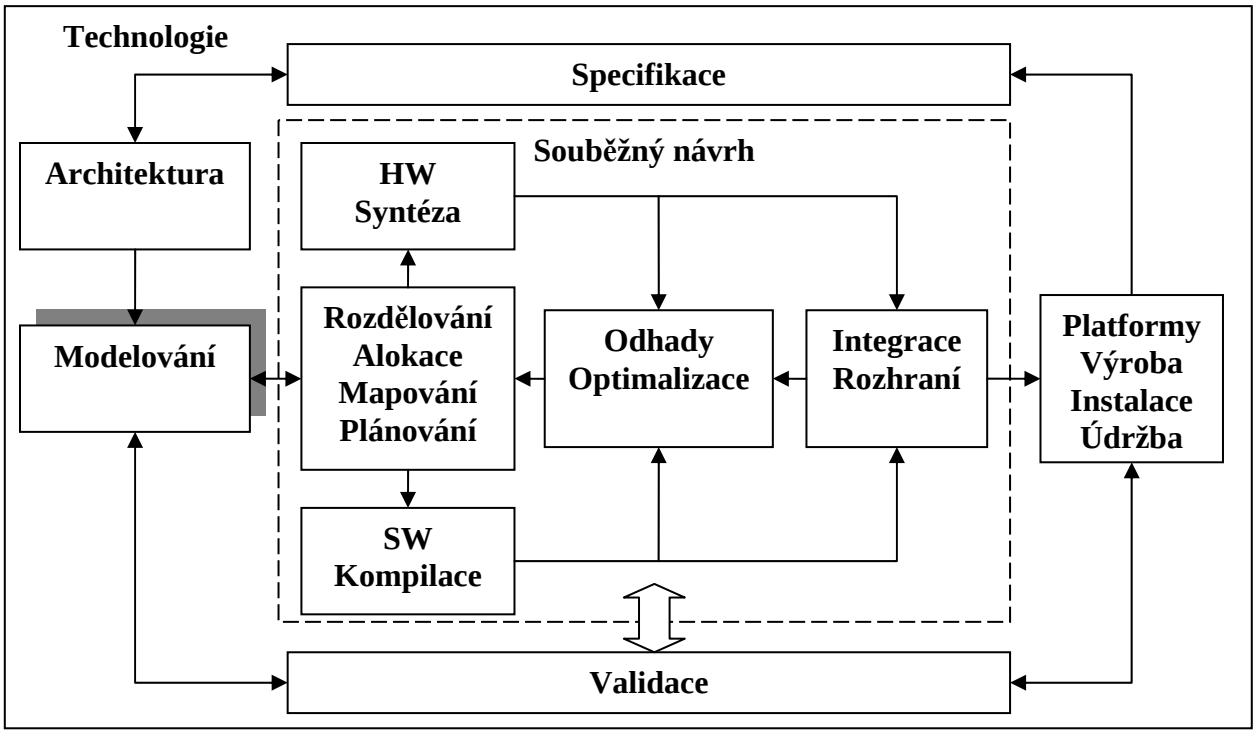
Obr. 18 Výuková platforma FITkit

4.10 SHRNUTÍ

HSC návrh výpočetních systémů je kompromisem mezi rychlostí jejich návrhu (a tedy i cenou) a efektivní implementací. Díky tomu, že počet tranzistorů roste rychleji, než produktivita práce návrháře stává se v dnešní době problémem jejich efektivní využití. Tato skutečnost je jedním z důvodů stále větší orientace na HSC a FPGA platformy, které jsou dostatečně flexibilní (rychlost návrhu je vyšší než při návrhu ASIC) a přitom dostatečně efektivní (využití tranzistorů je lepší než u MPU). Použití programovatelných výpočetních platforem umožňuje abstrahovat od fyzické výroby obvodu (podobně jako v případě MPU). Trendem je orientace na funkčnost a rychlý návrh systémů a ne na výrobu čipů. Zkušenosti s návrhu a použití platforem ve výzkumu, výuce a v komerčních aplikacích dokazují, že tento trend, poprvé autorem aplikován již v roce 1994, je správný a životaschopný i do budoucna. S ohledem na podstatu reálných aplikací představuje efektivní mapování reálných úloh na heterogenní a zároveň škálovatelné výpočetní struktury optimální cestu tvorby aplikačně specifických výpočetních systémů. S rostoucí složitostí platforem je však třeba klást stále větší důraz na vyšší úroveň abstrakce při jejich návrhu viz následující kapitola.

5 MODELOVÁNÍ

Složitost a heterogenní podstata dnešních výpočetních systémů je taková, že nevystačíme s jedním modelovacím nástrojem. Je třeba pracovat na co nejvyšší úrovni abstrakce a používat takové výpočetní modely, které nejlépe zohledňují charakter dané úlohy. Vhodně modelovaná úloha se též snáze a efektivněji implementuje. V této kapitole je tato problematika rozvedena s důrazem na modelování času a použití tzv. globálně asynchronních a lokálně synchronních obvodů (GALS). Kontext problematiky modelování při platformním HSC návrhu výpočetních systémů je znázorněn na Obr. 19.



Obr. 19 Modelování

Výpočetní model definuje vlastnosti systému a jazyk (např. C) popisuje svými výrazovými prostředky konkrétní výpočetní postup. Jeden jazyk lze typicky použít pro více účelů (modelování, simulaci, syntézu apod.). Jistý výpočetní model může být využit různými jazyky a určitý jazyk může podporovat více výpočetních modelů. Užívání určitého jazyka, ale i modelu či technologie, je však do značné míry dáno kulturním kontextem a historickými důvody[10] [69]. Pokud se většina inženýrů naučí ve škole programovat v jazyce C, bude jej v praxi s největší pravděpodobností též většina z nich používat a to i když mohou existovat lepší nástroje. Přechod na nový a možná v něčem lepší jazyk, ještě neznamená, že za pár let jej nebude třeba znovu změnit. Problémem je, jak často si takové změny můžeme dovolit provádět v celé komunitě programátorů. S rostoucími potřebami uživatelů se proto, spíše než přecházení na nové jazyky, v praxi provádí evoluce standardních jazyků.

Příkladem postupného vývoje použití jazyků pro různé účely jsou jazyky pro popis hardware VHDL a Verilog. Např. jazyk VHDL nebyl původně zamýšlen pro návrh HW, ale pro sjednocení popisu HW na kontraktech pro americké ministerstvo obrany. Od okamžiku standardizace VHDL jako dokumentačního jazyka je přirozené, že uživatelé jej chtěli využít i pro simulaci a syntézu.

[10] Viz tzv. Sapir-Whorf hypotéza, která říká, že jazyk, ve kterém se vyjadřujeme, ovlivňuje způsob, kterým přemýšlíme a děláme rozhodnutí.

První VHDL simulátory byly pomalé a nepodporovaly veškeré jazykové konstrukce. Jazyk Verilog byl oproti tomu od počátku navrhován jako simulační jazyk a představoval velký přínos pro efektivitu návrhu, neboť nabízel vše a právě jen to, co návrháři potřebují pro rychlé ověření návrhu simulací (dodnes je v USA standardem pro návrh ASIC obvodů).

Od podpory simulace je přirozený krok k využití jazyka pro syntézu. První syntezátory umožňovaly generovat jen malou množinu HW struktur a to jak v jazyce Verilog, tak VHDL. Postupem doby se tato množina rozšiřovala a rostla též kvalita syntetizovaného HW (syntezátory přesto dodnes nepodporují některé jazykové konstrukce). Používání HLD jazyků do jisté míry omezilo úroveň abstrakce při popisu HW, neboť i když dnešní HDL jazyky jsou schopny behaviorálního popisu, je popis na úrovni meziregistrových přenosů (angl. Register Transfer Level – RTL) dodnes de facto standardem (téměř 20 let) a možnostech modelování na systémové úrovni jsou velmi omezené. S ohledem na složitost dnešních výpočetních systémů, je však nezbytné modelovat HW na co nejvyšší úrovni abstrakce (viz např. modely KPN, CSP, SDF a CFSM uvedené v odstavci 5.2).

Řada konstrukcí, které je možno specifikovat v HDL jazyce, nelze efektivně syntetizovat do HW. Úspěšné využití HDL jazyků pro syntézu vyžaduje zkušenost, bez které je možné zapsat takový kód, ze kterého bude po syntéze velmi neefektivní HW.

Z důvodu určení HDL jazyků pro popis HW v nich nejsou některé jazykové konstrukce, známé z programování SW, podporovány. Typickým příkladem jsou např. ukazatele. Dnes je tento problém široce diskutován a trendem je orientace na využití jazyků, které jsou vhodné pro behaviorální popis (jako např. C).

V oblasti informačních technologií je většina lidí vyškolena pro zápis algoritmů v jazyce C. Problém však je skutečnost, že HW je paralelní ve své podstatě, kdežto zápis v jazyce C je sekvenční. Automatické generování paralelních struktur ze sekvenčního zápisu je netriviální problém. Existuje řada jazyků vycházejících z notace jazyka C, které byly adaptovány pro popis HW. Jedná se např. o jazyky Handel-C, SpecC, SystemC či Impulse-C. Autorovou zkušeností je, že použití podobných jazyků umožňuje často zvýšit produktivitu práce, ale typicky za cenu méně efektivních výsledků.

Nelze předpokládat, že bude k dispozici univerzální model, který by umožňoval modelovat veškeré aspekty komplexních výpočetních systémů. Je to dáno skutečností, že na jedné straně existuje obrovskému množství úloh různého charakteru a na druhé široké spektrum možných implementací. Existují snahy o vytvoření nových či rozšíření stávajících jazyků o další modelovací schopnosti, které budou zahrnovat co nejširší spektrum úloh. Tyto jazyky jsou však většinou používány pouze úzkou skupinou uživatelů a nepodporují proto řadu aplikačních domén. Např. jazyk SpecC [www.specc.org] představuje nadmnožinu jazyka C, který definuje příkazy pro nejvíce používané výpočetní modely (konečný automat atd.) či návrhové vzory (např. řetězené zpracování apod.), explicitně však nepodporuje modelování v aplikačních doménách (např. číslicového zpracování signálů). Slibným směrem je použití jazyka UML [41]. Existují i komerčně dostupné systémy pro modelování na vyšší úrovni abstrakce a následné automatizované generování jak kódu v jazyce C, tak VHDL a Verilog (např. prostředí Simulink firmy Mathworks).

Model je zjednodušeným popisem jiné entity, který odděluje podstatné od nepodstatného. Při návrhu můžeme přecházet z jednoho modelu ke druhému – obvykle od jednodušších popisů chování (algoritmus) ke složitějším strukturním popisům (výrobní předpis produktu). Model je tedy třeba volit s ohledem na ty vlastnosti, které v daném úkolu potřebujeme nezbytně vyjádřit a na ty, které můžeme zanedbat (např. s ohledem na urychlení simulace).

Využití komponent a platforem při návrhu dnes nabývá, díky složitosti dnešních výpočetních systémů, na důležitosti (viz kapitola 4). V praxi osvědčený přístup k HSC výpočetních systémů je založen na principu, který je společný všem inženýrským disciplínám kde platí, že komplexní systémy se nejsnáze staví z jednodušších, předem ověřených, komponent (subsystémů). Při tomto přístupu je vhodné oddělit popis výpočtů od popisu komunikace. Jednotlivé, obecně souběžně

(angl. concurrent), vykonávané výpočty (proces, vlákno) pak mezi sebou komunikují pomocí rozhraní (nejlépe standardizovaného) a jejich činnost je vzájemně synchronizována. Souběžností rozumíme nějakou kombinaci paralelizmu (vykonávání ve stejném čase na více výpočetních prostředcích) a prokládání (sdílení společných výpočetních prostředků v různém čase). Dekompozice modelu na souběžné procesy umožňuje použít různé popisy vhodné pro modelování každého z nich s ohledem na jeho charakter. Podobně jako souběžnost, tak i hierarchie popisu zlepšuje práci s komplexními modely chování, jež jsou rozděleny na jednodušší celky, se kterými lze odděleně a nezávisle pracovat.

Vzhledem k tomu, že komplexní praktické úlohy jsou ze své podstaty heterogenní, je jejich zpracování vhodné provádět na heterogenních výpočetních platformách (viz kapitola 4). Modelování heterogenních systémů může přinést lepší výsledky z hlediska efektivity výsledného systému, je však náročné na znalosti a čas. Jednou z možností, jak tento problém zjednodušit je využití nějakého unifikovaného modelu pro popis všech vlastností systému, což se však ukazuje jako téměř nereálné. Případně je možno spojit různé modely do jednoho. Výsledný model však bude složitý pro použití, syntézu a validaci. Další variantou je vybrat jeden model a ukázat, že ostatní jsou jeho speciálním případem. Teoreticky to lze (viz např. Turingův stroj), ale v praxi to bude, vzhledem k obtížnosti jeho použití, nepoužitelné. Konečně, asi nejlepší variantou je kombinovat modely dle charakteru heterogenních aplikací.

Mezi základní domény, ve kterých se při modelování pohybujeme, patří výpočty, data, paměť, komunikace a čas, vše na více úrovních abstrakce. Pro každou úroveň a doménu může být vhodné použít jiný způsob modelování (viz Tab. 3).

Tab. 3 Úrovně abstrakce a domény

	Výpočet	**Data**	**Komunikace**	**Čas**	**Paměť**
Omezení	Časová složitost	Datové typy	Šířka pásma, protokol	Latence, propustnost	Paměťová složitost
Model	Algoritmus	Reprezentace	Sdílená paměť, zasílání zpráv	Kauzalita, synchronní, atd.	Hierarchie
SW	ISA	Operandy	Volání procedur, přerušení	Instrukční cyklus	Adresovací režimy
HW	Kombinační a sekvenční obvody	Logické úrovně	Rozhraní, propojení na čipu, PCB, šasi	Hodinový signál	Organizace, doba přístup, doba cyklu
Implementace	Tranzistory	Elektrické signály	Vodiče	Zpoždění	SRAM, DRAM, disk

Především čas představuje doménu, která je, na rozdíl od tvorby běžného SW, nezbytnou součástí každého návrhu HW. Obdodně, v případě HW se typicky neuvažuje o složitých datových strukturách, které vyžadují např. ukazatele, což je jeden z často používaných mechanizmů při tvorbě SW. Při HSC návrhu heterogenních systémů se však musí zohlednit všechny aspekty HW i SW.

V kontextu hybridních výpočetních platforem se zabýváme modelováním souběžných výpočetních úloh, jejich komunikací, synchronizací, časováním a sdílením výpočetních prostředků. Alokace, mapování a plánování výpočetních zdrojů jednotlivým vláknům jsou základními problémy procesu rozdělování, které se snaží optimalizovat HSC (viz kapitola 7).

V literatuře zabývající se HSC systémů je popsána řada přístupů k problematice modelování komplexních heterogenních systémů pomocí různých modelů. Nejčastěji se jedná o kombinaci modelů v rámci vývojového prostředí. Příkladem je heterogenní modelovací prostředí Ptolemy [37], ve kterém přidání nového modelu vyžaduje definici rozhraní na veškeré ostatní modely, které již dané prostředí obsahuje tak, aby byla zajištěna jak komunikace mezi nimi, tak jejich vzájemná synchronizace. Při evaluaci tohoto prostředí se ukázalo, že je vhodné pro výzkum v dané oblasti, ale vzhledem k složitosti se jeho použití v praxi jeví jako omezené. Dalším přístupem k problematice koexistence heterogenních modelů je vytvoření návrhového prostředí, které představuje jakési komunikační médium zajišťující přenos dat mezi modely. Příkladem tohoto přístupu je vyvinutý cosimulační systém, který je popsán v kapitole 8.1.

Vzhledem k šíři uvedené problematiky se zde omezíme pouze na modely, které byly aplikovány při návrhu komplexních vizuálních systémů (viz kapitoly 12 a 13) s výpočetními platformami na bázi FPGA (GALS obvody viz dále).

5.1 MODELOVÁNÍ ČASU

Při tvorbě systémů pracujících v reálném prostředí představuje čas jednou z nejdůležitějších domén. Podle způsobu práce s časem si modely rozdělme na kauzální (čas zohledňují ve formě příčinnosti – stavy a přechody), synchronní, hodinově synchronní a na modely s explicitně vyjádřeným časem. Reprezentace času hraje na všech úrovních abstrakce podstatnou roli a způsob práce s časem rozlišuje jednotlivé modely od sebe. Uveďme si základní vlastnosti modelů s ohledem na čas, viz Tab. 4:

Tab. 4 Model a čas

Model času	**Čas**	**Výpočet**	**Komunikace**
Výkonnost	Latence	[s]	[s]
	Propustnost	$[b \cdot s^{-1}]$	$[b \cdot s^{-1}]$
Kauzalita	Uspořádání událostí	∞ s	∞ s
Synchronní	Událost	0 s	0 s
Hodinově synchronní	Periodická událost (hodinový signál)	Δt [s]	0 s
Fyzický	Zpoždění	[s]	[s]

Výkonnost systému je dána dobou zpracování úlohy (latence) a četností, s jakou systém úlohy zpracovává (propustnost). Vlastnosti komunikačního média jsou dány zpožděním přenosu signálu (latence) a množstvím dat přenesených za jednotku času (propustnost a šířka pásma). Při návrhu systémů je výkonnost definována ve formě omezující podmínky.

Kauzuální modely

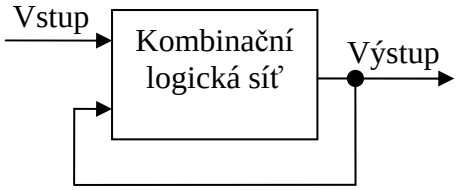
V kauzálních modelech je čas přítomen formou uspořádání událostí (příčina – následek) a jak výpočet, tak komunikace jsou provedeny za nedefinovanou dobu. Příkladem jsou tzv. dataflow sítě procesů (např. KPN, CPS, SDF, viz odstavec 5.2), ve kterých mezi sebou procesy komunikují pomocí signálů. Signály přenáší data mezi producentem a konzumentem, zachovávají jejich pořadí, ale nenesou žádnou informaci o čase. Dataflow sítě mají vlastnosti, které usnadňují modelování na nejvyšší úrovni abstrakce. Popisují vzájemné propojení a interakci procesů, nemodelují však procesy samotné. Implementace procesů je pak závislá na použitém jazyce pro jejich popis a na následné syntéze (HW) či kompilaci (SW). Díky svým vlastnostem jsou tedy dataflow modely vhodné pro popis hybridních systémů sestavených za použití různých výpočetních architektur a technologií.

Synchronní modely

V synchronních modelech je časová osa rozdělena na uspořádané sloty, které jsou tvořeny synchronizačním událostmi. Komunikace i výpočet proběhnou okamžitě po příchodu synchronizační události. Čas je zde přítomen nepřímo tím, že vstupní události jsou úplně uspořádány. Model je vhodný pro popis systémů, které na základě detekce události provedou výpočet a na výstup odešlou vypočtenou hodnotu ještě před příchodem nové události. Výhodou je, že při simulaci bude chování systému stejné (dobu výpočtu i komunikace můžeme zanedbat) jako v reálné aplikaci. U těchto modelů je odděleno časové od funkčního chování, což zjednodušuje modelování. Typicky to znamená, že se návrhář může soustředit na specifikaci funkce a její validaci a má jistotu, že systém bude pracovat správně, pokud implementace bude dostatečně rychlá.

V synchronních modelech se předpokládá, že výpočet i komunikace trvají nulovou dobu. Výstup je generován na základě vstupní události a je ihned globálně platný. Hovoříme o reaktivním chování, kdy systém musí reagovat tak rychle, jak to okolní prostředí vyžaduje. Příkladem je směrování paketů v počítačových sítích, kdy se po příchodu paketu, musí provést směrování a odeslání paketu ještě před příchodem paketu nového. Při implementaci však bude mít synchronní obvod chování, které bude dáno jeho fyzikálními vlastnostmi, neboť jak výpočet, tak komunikace netrvají nekonečně krátkou dobu.

Vlastnosti modelu s ohledem na čas si s ohledem na orientaci na návrh platforem s FPGA dále uveďme na příkladu popisu logických obvodů. V logických obvodech je trvání výpočtu dáno zpožděním logických členů a spojů v logických sítích a komunikace je určena zpožděním na vstupních a výstupních vodičích. Výstup logického obvodu tedy bude platný až za jistou dobu. Pokud u synchronního modelu zavedeme zpětnou vazbu, viz např. kombinační logická síť (KLS) na Obr. 20, dostáváme po jeho fyzické implementaci logický obvod – tzv. asynchronní sekvenční síť. Návrh asynchronních obvodů je poměrně náročný, neboť jejich chování je ovlivněno jejich fyzikálními vlastnostmi, které se mění s pracovní teplotou, napájecím napětím atd. Po příchodu vstupní události se na výstupu mohou objevovat různé posloupnosti, hodnot, které jsou dány různou dobou šíření signálů obvodem (tzv. hazardy).



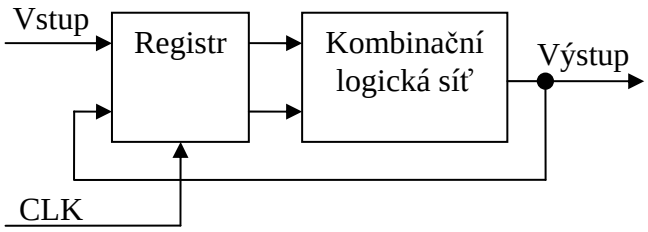
Obr. 20 Synchronní obvod se zpětnou vazbou

V praxi si proto jejich návrh zjednodušujeme vhodným omezením režimů jejich činnosti (tzv. fundamentální či impulsní režim), kdy se vždy přivádí vstupní událost jen na jeden vstup a počká se na ustálení výstupů obvodu, než se může aplikovat další událost (pracuje tedy navenek synchronně, interně je však asynchronní). Pokud dodržíme uvedená omezení, lze implementovat užitečné asynchronní obvody (např. klopné obvody.

Hodinově synchronní modely

Pokud umístíme klopné obvody před vstup KLS synchronního obvodu, dostaneme tzv. hodinově synchronní obvod. U tohoto obvodu je činnost řízena událostmi na vstupu CLK, které jsou generovány nejčastěji periodicky pomocí tzv. hodinového signálu. Výstup obvodu se zpětnou

vazbou je tedy platný až po příchodu následující synchronizační události (v následujícím hodinovém cyklu). Díky tomuto zjednodušení (za cenu složitějšího a tedy dražšího obvodu) se výrazně usnadňuje návrh sekvenčních logických sítí, které lze též snadno automatizovaně syntetizovat (nižší cena návrhu). Z tohoto důvodu dnes hodinově synchronní modely dominují při návrhu číslicových systémů. Pro dodržení předpokladu, že výpočet má nulové trvání, stačí při implementaci zajistit, že perioda hodinového signálu je delší než je zpoždění nejdelší logické větve KLS. Hodinově synchronní obvodu se snadno implementují a modelují, např. pomocí konečných automatů. Příklad hodinově synchronního logického obvodu vytvořeného z registru a KLS je na Obr. 21.



Obr. 21 Hodinově synchronní obvod

Typicky je pak v celém obvodu rozveden jeden hodinový signál a všechny hodinově synchronní obvody jsou na něj synchronizovány. Toto však často vede na situaci, kdy KLS v některých částech obvodu může být rychlejší (kratší větve KLS) a není tedy efektivně využita (daný obvod může pracovat na vyšších frekvencích). Dalším problémem je zajištění synchronnosti hodinového signálu v celém systému. Díky rychlosti šíření elektrických signálů a délkám spojů často velmi obtížné zajistit, aby všechny klopné obvody reagovaly na hranu hodinového signálu ve stejný okamžik.

Hodinově synchronní modely představují z hlediska časové domény kompromis mezi kauzálním a časovanými modely. Většina detailů implikovaných zavedením času je v nich ignorována, přesto je zde však čas, ve snadno simulovatelné a syntetizovatelné formě, přítomen díky synchronizaci. Hodinově synchronní modely též leží někde uprostřed mezi čistě behaviorálním vyjádřením chování systému a jeho strukturním popisem. Příkladem je syntéza RTL modelu v jazyce VHDL, ve kterém je synchronizace zavedena pomocí tzv. atributů. Např. instance klopného obvodu citlivého na vzestupnou hranu hodinového signálu bude vygenerována na základě následující jazykové konstrukce:

```
process(CLK) -- synchronizační událost
begin
  if (CLK='1')and(CLK'event)then – kladná hrana
     Q <= D;
  end if;
end process;
```

Uvedený zápis definuje, že se má na výstup Q obvodu kopírovat logická hodnota, která je na vstupu D v době přechodu signálu CLK do hodnoty log. 1. Atribut „event“ detekuje změnu hodnoty signálu CLK. Protože po této změně je jeho hodnota rovna log. 1, pak v reálném a správně pracujícím systému, musela do ní přejít z hodnoty log. 0. Při simulaci tomu tak být nemusí, neboť do log. 1 lze přejít i z jiných hodnot než log. 0 (dáno typem signálu). Po syntéze bude obvod používán jako hodinově synchronní obvod, který byl modelován jako synchronní obvod se zpětnou vazbou a který interně pracuje asynchronně.

To, že obvod bude pracovat, jak bylo výše uvedeno, je dáno dohodnutou interpretací výrazu „`(CLK='1') and (CLK'event)`“. Snadný návrh hodinově synchronních obvodů však vede na nízkou efektivitu jejich implementace (příkon, výkonnost).

Časované modely

Při implementaci modelů je však vždy nezbytné uvažovat čas (zpoždění) jako základní vlastnost každého fyzicky realizovaného obvodu. V řadě případů musí být čas uvažován s velkou přesností a měřen v časových jednotkách. Chování systému v čase ovlivňuje i jeho funkční chování (např. pokud je odezva systému pomalá, nemusí zpracovat všechny vstupní data a jeho chování bude nedeterministické).

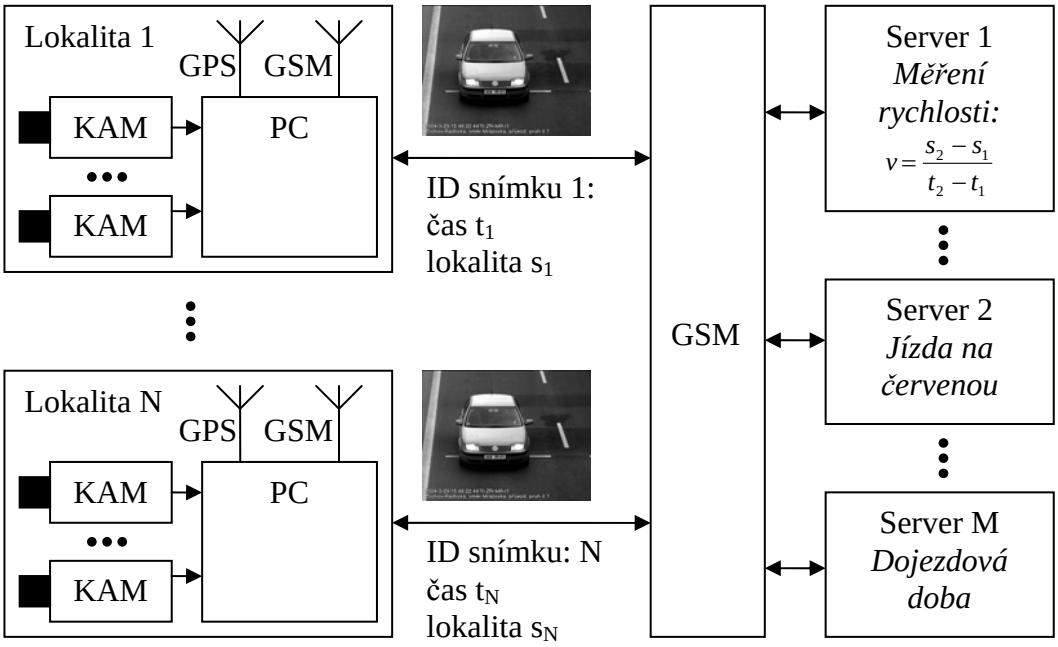
V časových modelech se přiřazuje časové razítko každé hodnotě, která je komunikována mezi procesy, což umožňuje modelovat časově závislé chování, avšak výrazně komplikuje analýzu a syntézu systému. Např. v hodinově synchronním modelu je časová informace distribuována signály jako události, které jsou generovány v určitých časových intervalech (hodinový signál). Rozdíl časovaného od hodinově synchronního modelu je v tom, že zrnitost času je zde mnohem jemnější. Události vznikají např. každou nanosekundu a ne jen s hodinovým signálem (např. 1× za 100 ns). Konkrétní jednotka fyzického času je záležitostí interpretace. Teoreticky je jedno, zda model pracuje s jednotkou ns či ps, prakticky je však nutné, aby tyto jednotky byly v souladu s konečnou implementací, která je dána použitou cílovou technologií a požadavky na chování reálného systém. Je tím též dána rychlost simulace, kdy procesy, které mají pracovat s odezvou v řádu milisekund, nemá smysl simulovat s přesností na pikosekundy.

Časované modely nelze syntetizovat, neboť časová informace vychází z vlastností použitých komponent. Jedná se o dobu výpočtu, která je dána jednak použitou technologií a architekturou výpočetního prostředku. A dále pak zpožděním při komunikaci, která je určeno použitým komunikačním médiem a protokolem. Tato časová informace je důležitá při simulaci, neboť umožňuje simulovat jak funkční, tak i časové chování modelu před jeho výrobou. S ohledem na syntézu je však časová informace nadbytečná, neboť prakticky nelze provádět syntézu na základě definice časového chování komponent.

Typicky se tedy syntéza provádí na základě funkční specifikace chování komponent. Pokud např. bychom v modelu definovali, že konkrétní logický člen AND má mít zpoždění 1 ns, pak syntezátor bude informaci o čase ignorovat a vygeneruje seznam spojů, ve kterém bude vložen logický člen AND z knihovny předefinovaných komponent pro danou cílovou technologii. Syntezátor nebude navrhovat logický člen AND se zpožděním právě 1 ns, protože to nelze ve výrobě dodržet. Můžeme samozřejmě dojít k případu, kdy knihovní prvek bude mít za určitých podmínek (velikost napájecího napětí, teplota atd.) zpoždění právě 1 ns, ale typicky to bude tak, že se použije prvek, který bude mít zaručené zpoždění pod 1 ns za všech pracovních podmínek. To, že se vybere prvek se zpožděním pod 1 ns, však nebude řízeno časovou informací z časovaného modelu, ale doplňkovou informací pro syntezátor, kterou se řídí proces syntézy. Každý syntezátor má možnost definice omezujících podmínek optimalizace výsledné implementace, typicky s ohledem na čas (např. minimální frekvence synchronizačních hodin) a plochu (např. minimalizace logických výrazů s ohledem na co nejmenší počet logických členů). Podobně v případě komunikace mezi komponentami (např. vodiči mezi logickými členy) je, s ohledem na množství a omezenou plochu, prakticky nemožné navrhovat spoje s konkrétní délkou, která přesně definuje jejich zpoždění. Při syntéze se pak informace o čase ignoruje a model se interpretuje a syntetizuje jako hodinově synchronní model.

Nejčastěji se časované modely implementují pomocí časovače či časových razítek. Příkladem je systém Unicam (viz [22] a [75] a kapitola 12), ve kterém byl použit časovaný model pro globální synchronizaci událostí. Díky tomu může být systém prostorově dekomponován. V jednotlivých lokalitách jsou umístěny inteligentní kamery a výpočetní platformy (viz odstavec 4.8), které jsou

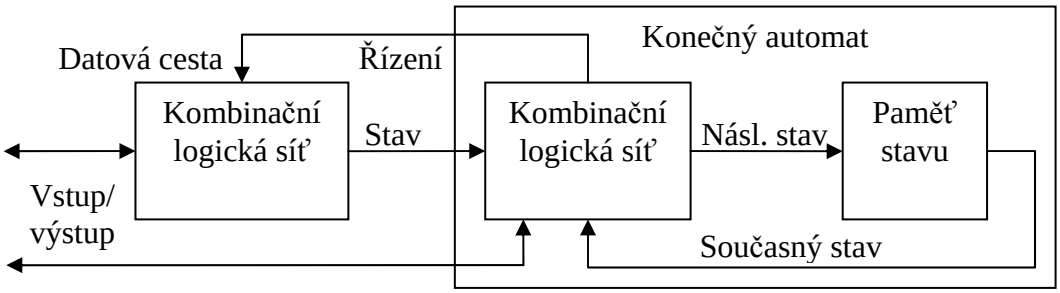
synchronizovány na GPS čas s přesností na 1 ms (uvedená přesnost je jednou z unikátních vlastností systému). Jak časová informace, tak informace o umístění kamer je součástí identifikace snímků vozidel pořizovaných systémem. Tímto je umožněna tvorba různých aplikací, které ve svých algoritmech zohledňují místní a časovou informaci. Jedná se např. měření průměrné doby jízdy mezi dvěma lokalitami atd. (více viz kapitola 12).



Obr. 22 Čas v systému Unicam

5.2 GLOBÁLNĚ ASYNCHRONNÍ A LOKÁLNĚ SYNCHRONNÍ MODELY

Tradičním způsobem popisu HW výpočetních systémů je kombinace dataflow (kombinační logická síť) a controlflow modelů chování (hodinově synchronní konečný automat).



Obr. 23 Konečný automat s datovou cestou

Při modelování řady praktických úloh však s výše uvedeným modelem nevystačíme. Složité systémy jsou typicky rozděleny do menších, souběžně pracujících a komunikujících procesů. Při implementaci takových systémů se osvědčilo sestavovat je z obvodů, jež jsou hodinově synchronní a mezi sebou komunikují komunikačním mechanizmem, který je vzhledem k jejich činnosti asynchronní (tzv. globálně asynchronní a lokálně hodinově synchronní – GALS obvody) [64].

Nejčastěji používaný přístup je takový, že jednotlivé procesy jsou navrženy jako hodinově synchronní obvody a komunikace mezi nimi je realizována pomocí asynchronního zasílání zpráv s (1) „rendez-vous“ mechanizmem (CSP), (2) přes sdílené registry (CFSM) či (3) přes fronty FIFO (KPN) viz dále. Použití GALS obvodů má mnoho praktických výhod. Např. u systémů dekomponovaných mezi různé integrované obvody na deskách s plošnými spoji je obtížné zajistit synchronní komunikaci (různé zpoždění vodičů). Jednotlivé obvodu mohou interně pracovat jako hodinově synchronní, jejich vzájemná komunikace však zohledňuje zpoždění vodičů (je asynchronní). Důsledkem je to, že každý lokálně hodinově synchronní obvod může pracovat na své maximální frekvenci, což se příznivě projeví na rychlosti celého systému a zároveň lze dosáhnout omezení příkonu takového obvodu.

U GALS obvodů se s výhodou využívají předem ověřené komponenty, vždy je však třeba provést validaci celého obvodu, neboť např. producent může generovat výstupy rychleji než je konzument schopen je akceptovat a chování systému pak může být nedeterministické (např. CFSM, KPN). Chování systému je též časově závislé díky tomu, že asynchronní komunikace může obecně trvat různou dobu.

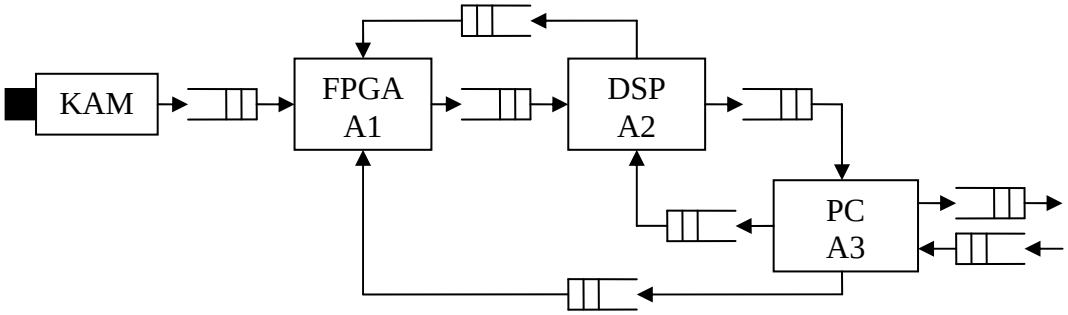
Pro modelování GALS systémů je možno použít řadu přístupů. Zde si uveďme ty, které byly ověřeny při tvorbě komplexních výpočetních systémů na úrovni platforem s FPGA, desek s plošnými spoji a celých systémů. Rozdělením úlohy na komunikující procesy je usnadněn HSC systému je a jeho implementace heterogenním způsobem na platformách s FPGA.

Kahnova síť procesů (KPN)

Asi nejobecnějším modelem vhodným pro modelování GALS systémů je tzv. Kahnova síť procesů (angl. Kahn Process Network – KPN) [63]. KPN je tvořena sekvenčními procesy, které mezi sebou komunikují zasíláním zpráv. Komunikace probíhá přes kanály, které jsou implementovány pomocí FIFO front s teoreticky neomezenou kapacitou. Producent může do FIFO psát kdykoliv (neblokující zápis) a konzument může z FIFO fronty číst jen, pokud jsou v ní data – je tedy blokován, pokud nemá na vstupu data (blokující čtení). Tímto je zajištěno deterministické chování, neboť proces musí počkat na data a nemůže např. testovat, zda je FIFO prázdná a případně (pokud nejsou k dispozici data) pokračovat ve výpočtu. Tím by mohlo docházet k situaci, kdy čtení z FIFO by probíhalo v závislosti na stavu výpočtu procesu a ne na přítomnosti dat na jeho vstupech a chování by mohlo být nedeterministické. Díky předpokladu, že fronty mají neomezenou kapacitu, je KPN model ekvivalentní s Turingovým strojem [53].

Při implementaci však vždy platí, že fronta FIFO použitá pro komunikaci je omezené velikosti a obecně tedy nelze rozhodnout, zda výpočet skončí v konečném čase. Model může uvíznout, pokud procesy nemají data ke zpracování, či nestačí omezená hloubka FIFO pro uložení všech dat čekajících na zpracování [5].

Např. v systému Unicam (viz kapitoly 12 a 13) je pomocí KPN modelován tok dat mezi kamerou, výpočetní platformou DX6 s FPGA a DSP, vyhodnocovacím počítačem a centrálním serverem (algoritmus detekce vozidla v zorném poli kamery je rozdělen do 3 stupňů (A1, A3 a A3) viz odstavec 5.3). Pro eliminaci možného uvíznutí díky omezené velikosti front FIFO je v systému Unicam jednak provedena statická analýza toku dat v systému, takže velikost front je předem kvalifikovaně odhadnuta. Dále je zabudován systém on-line kontrol aktuálního stavu front FIFO. Pokud dojde k jejich přeplnění, je tato skutečnost zaznamenána a případně vyvoláno přerušení, při kterém se uvedená výjimka definovaným způsobem ošetří, viz např. odstavec 8.3. Dalším zdrojem možného nedeterminismu je situace, kdy komunikační kanál zahazuje data. Porucha přenosu dat je opět v praxi možná a v nějaké formě přítomná vždy. K uvíznutí pak dojde v případě, že kanálem data nedošla ne proto, že je producent neodeslal, ale že je kanál nepřenesl. Uvedené situace se ošetřují aplikačním protokolem (potvrzováním).



Obr. 24: Kahnova síť procesů v systému Unicam

KPN model přirozeně a efektivně modeluje souběžnost v aplikacích s datovými proudy (díky frontám FIFO, do kterých lze zasílat proudy dat a producent dat není blokován, než si konzument data převezme).

Synchronní dataflow model (SDF)

Pokud omezíme modelovací schopnost modelu tak, aby otázky velikosti FIFO front byly předem rozhodnutelné, lze vyřešit problém s nedetrminismem. Příkladem je tzv. synchronní[11] dataflow model (angl. Synchronous Dataflow – SDF) [36]. V SDF modelu je pro spuštění procesu třeba, aby ve vstupní frontě byl určité množství dat. Podobně je dán objem dat, který je při vykonání procesu produkován. Pokud předpokládáme, že topologie sítě je pevně daná a že též objem dat produkovaných a konzumovaných procesy je fixní, pak lze případné uváznutí odhalit a velikost FIFO front stanovit předem. SDF modely jsou tedy deterministické, jejich modelovací schopnost je však omezena na úroveň konečných automatů.

SDF se používá nejčastěji pro modelování algoritmů číslicového zpracování signálů, u kterých probíhá výpočet typicky jako stále se opakující vykonávání výpočetních operací na nekonečném toku dat. Aplikace musí pracovat v reálném čase a je nezbytné, aby výpočet byl deterministický. Dalším důvodem pro použití SDF v doméně DSP je jeho podobnost s blokovými diagramy, které se využívají pro návrh a dokumentaci DSP algoritmů. V platformách s FPGA jsou pomocí SDF modelovány algoritmy číslicového zpracování obrazu.

Komunikující sekvenční procesy (CSP)

Komunikující sekvenční procesy (angl. Communicating Sequential Processes – CSP) [27]. Na rozdíl od KPN je CSP model deterministický (je ekvivalentní s FSM), neboť nekomunikuje přes neomezenou frontu FIFO, ale využívá jednoduchého a robustního mechanizmu zasílání zpráv pomocí mechanizmu „randez-vous“ (synchronní komunikace, blokujících čtení a zápis, „handshake“ protokol).

CSP model se využívá u řady komerčních návrhových systémů pro HSC systémů s FPGA. Autor má dobrou zkušenost s návrhovým systém od firmy Impulse [www.impulsec.com]. Tento systém umožňuje unifikovaný popis aplikace v jazyce Impulse-C, automatickou syntézu HW (např. VHDL) a kompilaci SW (např. pro MPU v FPGA). Podobné vlastnosti má i návrhový systém firmy Codetronix (využívá jazyk Mobius) [www.codetronix.com] či firmy Celoxica (jazyk Handel-C) [www.celoxica.com]. Klasickým jazykem s CSP výpočetním modelem je Occam [51].

[11] Model je spíše „statický“ než „synchronní“ – název je použit pro odlišení od statického plánování vykonávání modelu.

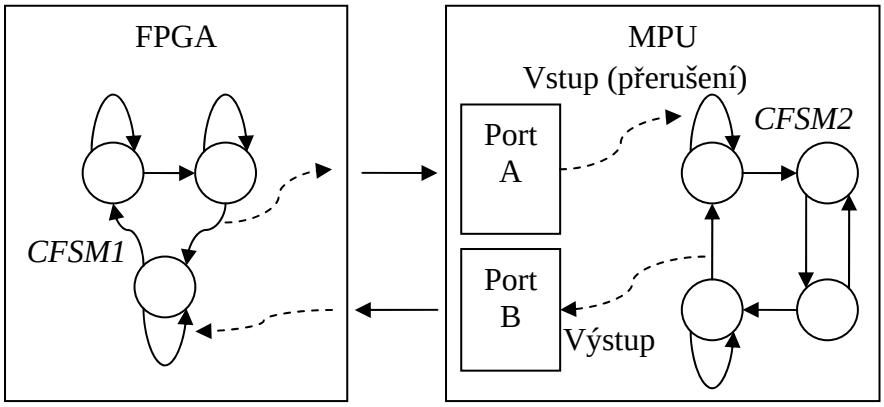
CSP modelování GALS obvodů bylo použito při návrhu FPGA při implementaci subsystémů inteligentní kamery systému Unicam (viz kapitoly 12 a 13). Zde bylo provedeno manuální propojení jednotlivých hodinově synchronních obvodů s datovými cestami z toho důvodu, že byly též aplikovány techniky adaptace frekvence hodinového signálu na základě provádění výpočtů (více viz odstavec 6.2), neboť to není dosud podporováno automatizovanými nástroji.

Codesign konečný automat (CFSM)

Dalším v praxi osvědčeným GALS modelem jsou tzv. codesign konečné automaty (angl. Codesign FSM – CFSM) [28], ve kterém mají procesy (hodinově synchronní Moorovy konečné automaty) na vstupu registr. Komunikace probíhá pomocí signálů, které nesou informaci o spouštěcí události a k ní přiřazených datech. Komunikace probíhá asynchronně a má nedefinované zpoždění. Producent může komunikovat s více konzumenty a může též přepsat data, která si konzument nestihl přečíst. Konzument čeká na událost na kterémkoliv ze vstupů, přečte data a provede příslušnou činnost a generuje výstup. Podmínkou deterministického chování je, že implementace procesů musí být dostatečně rychlá, aby nedocházelo k přepisování dat producentem.

Rozdíl mezi CSP a CFSM je ve způsobu komunikace a synchronizace procesů. CSP používá randez-vous komunikaci kde není pro data alokováno žádné úložiště (jako je FIFO či sdílená paměť) a zápis a čtení probíhá ve stejném čase. CFSM používá neblokujícího zápisu a blokujícího čtení, podobně jako KPN, avšak přes registr (FIFO hloubky 1).

Model byl aplikován např. při komunikaci mezi MPU a FPGA v inteligentní kameře systému Unicam. V MPU jsou jednotlivé procesy implementovány jako obsluha příslušného přerušení. Přerušení představuje událost, která iniciuje spuštění příslušného procesu, jenž si přečte data ze vstupní periferie, provede výpočet a uloží jej na adresu, na které jsou mapovány registry FPGA. Zápisem do registrů se spustí vykonávání příslušného konečného automatu s datovou cestou. Ilustrace komunikace mezi jednotlivými CFSM je na Obr. 25.



Obr. 25: Codesign konečné automaty

5.3 NÁVRH ALGORITMŮ

Při tvorbě systémů lze vybírat z řady algoritmů vhodných pro řešení konkrétních problémů (např. pro zpracování obrazu). V praktických aplikacích většinou není možné použít algoritmus, který byl ověřen na nějaké příkladové studii a syntetických problémech. Např. videodetekční algoritmy musí pracovat za všech světelných podmínek (den, noc a přechody mezi nimi), což klade enormní nároky nejen na návrh algoritmu, ale též na jeho testování v reálných podmínkách. Dále pak algoritmus musí být efektivně implementovatelný s ohledem na dostupné výpočetní zdroje a práci v reálném čase.

Návrh prakticky použitelných algoritmů musí adresovat řadu otázek souvisejících s výkonností algoritmů, studiem vhodných výpočetních prostředků a datových struktur. Jde o kombinaci teoretických metod spolu s důkladným experimentálním zkoumáním. Zkušeností ukazují, že experimentování na velkých souborech reálných dat je nejdůležitějším krokem při návrhu a analýze algoritmů, neboť to umožňuje ověření výchozích předpokladů a vede k přiblížení otázek návrhu algoritmu blíže k problému, kterým byl původně vývoj algoritmu motivován.

Např. algoritmy pro vizuální systém Unicam byly testovány na desítkách tisíc snímků pořízených za všech podmínek reálného provozu (noc, den, přechody mezi nimi). Při návrhu uvedeného systému bylo např. třeba vyvinout algoritmus pro detekci registrační značky (RZ) vozidla nacházejícího se v zorném poli kamery. Původně se předpokládalo, že se použije vhodný algoritmus pro zjištění spektra (přítomnost vyšších frekvencí v obraze jsou jedním z příznaků znaků RZ) jako je např. rychlá Fourierova transformace (FFT) a že výpočet poběží na PC. Při analýze problému se však zjistilo, že jen pro přenos všech snímků mezi kamerou a PC je třeba velké části (až 40 %) výpočetního výkonu PC (TCP/IP protokol), což bylo vzhledem k možnostem dostupné infrastruktury neakceptovatelné (omezený výpočetní výkon PC, kapacita přenosových tras, příkon, atd.). Proto se přistoupilo k návrhu inteligentní kamery UnicamD (viz odstavec 4.8), do které byla vestavěna výpočetní platforma DX6 (viz odstavec 4.3) obsahující FPGA, DSP a rozhraní Ethernet 100 $Mb \cdot s^{-1}$. Tímto se otevřela možnost realizovat část výpočtu daného algoritmu přímo v kameře. Umožnilo to selekci snímků, na kterých jsou jen vozidla pro jejich další zpracování (např. automatické čtení RZ). Z kamery se nemusí transferovat všechny generované snímky, což je, vzhledem k omezené šířce přenosového kanálu, nemožné. V kameře (platforma DX6 s DSP a FPGA) běží část algoritmu a provádí se primární filtrace zájmových snímků (přítomnost vyšších frekvencí v oblastech zájmu). Parametry této části algoritmu jsou nastaveny tak, aby pravděpodobnost toho, že se vozidlo detekuje, byla větší, než že nedetekuje (raději se detekují i neplatné vzruchy, než aby se vozidla nedetekovala). Vybrané snímky se posílají na další zpracování do PC, kde se z filtrovaných snímků vyberou jen ty, na kterých je opravdu vozidlo s RZ. Zde se zpracovává podstatně méně snímků (cca 1/10), což umožnilo použít sofistikovanější algoritmy. Výsledný algoritmus je rozložen na celkem 3 různé výpočetní prostředky (FPGA, DSP a PC). Díky uvedené dekompozici má systém následující vlastnosti:

- lze použít méně výkonné PC,
- systém má nižší příkon a lze jej nyní napájet z baterií,
- systém má rezervu ve výpočetním výkonu, což umožňuje budoucí doplnění nových algoritmů,
- je potřebná menší šířka pásma komunikačního kanálu mezi kamerou a PC (posílají se jen zájmové snímky a ne všechny),
- systém má nižší úroveň rušení.

Výše uvedený postup byl zobecněn a pro potřeby měření doby jízdy vozidel (viz kapitola 12) patentován [6].

Na výše uvedeném příkladě je demonstrován přístup k HSC komplexních systémů, kdy se nejedná jen o rozdělení výpočtu aplikace mezi SW a HW, ale též o zpětné ovlivnění specifikace a implementace úlohy na základě možností dostupných technologií (viz platforma DX6). Výsledkem je kvalitnější produkt (rychlejší, levnější, s nižším příkonem ap.) a možnost doplnění nových funkcí (rezerva ve výpočetním výkonu).

Dalším příkladem optimalizace algoritmů je návrh obrazových filtrů, který byl již od počátku zaměřen na maximální možné využití výpočetních struktur obvodů FPGA ve výpočetních platformách. Jedná se např. o morfologické filtry a akcelerátory klasifikátorů implementované v HW [4], [9]. Proces extrakce parametrů a klasifikace obrazu pro potřeby detekce znaků v obraze [8] je ilustrován na Obr. 26. U těchto algoritmů byl již od počátku požadavek na jejich efektivní implementaci v HW. Výhodou použití FPGA platforem je skutečnost, že je možno algoritmy vyvíjet v reálném systému. Jednotlivé iterace návrhu se testují na reálných datech, za provozu

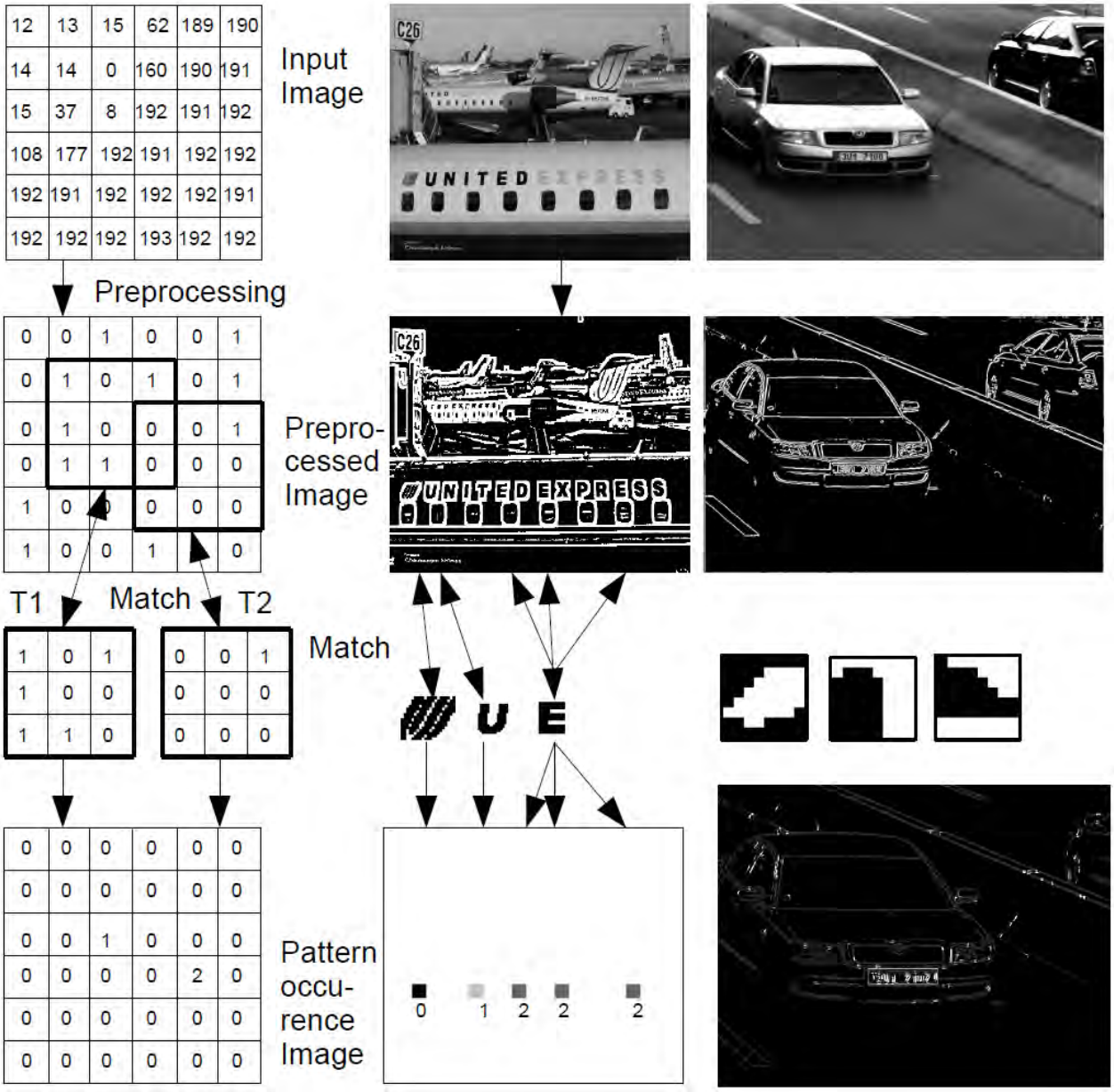
systému. Tímto se dosahuje značného urychlení vývoje, neboť se jednotlivé iterace návrhu nemusí simulovat, ale ověřují se na plné rychlosti systému. Dále se provádí neustálá optimalizace vyvinutých algoritmů, kdy se za provozu sbírají statistiky o účinnosti videodetkčních algoritmů a po jejich vyhodnocení se provádí jejich adaptace.

5.4 SHRNUTÍ

Heterogenní výpočetní systémy mají charakter (1) transformační, kdy systém počítá výstupy ze vstupů, (2) interaktivní – systém pracuje na základě dat z okolního prostředí, které generuje nové požadavky až po přijetí výstupu z právě probíhajícího výpočtu a (3) reaktivní, při kterém musí být výstup systému vypočten před příchodem nového vstupu (pracuje tedy na rychlosti okolního prostředí). Každou z uvedených vlastností je obecně vhodné modelovat jiným způsobem.

Výběr toho nejvhodnějšího modelu pro danou úlohu a jeho validní zápis představuje nejsložitější a nejdůležitější krok při návrhu systémů, který je vždy tvořen člověkem a nelze jej zautomatizovat.

Jedním z důvodů, proč je třeba používat více modelů, je též skutečnost, že systémy jsou dnes v převážné míře heterogenní, sestavené ze subsystémů určených pro specifické funkce. Heterogennost je dána tím, že tyto systémy jsou často sestaveny z různých výpočetních prostředků (MPU, DSP, FPGA ap.), jsou zasazeny do reálného prostředí (analogové vstupy a výstupy) a realizují často velmi různorodé funkce.



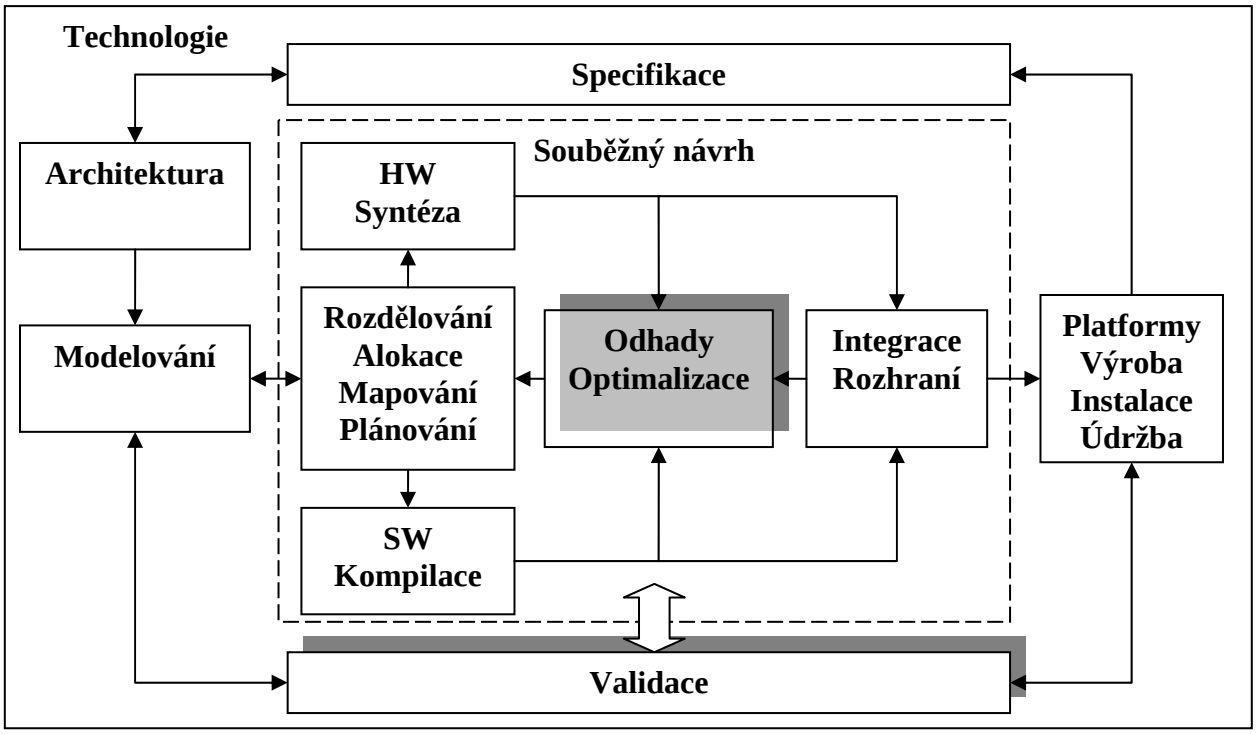
Obr. 26 Detekce znaků v obraze implemetovaná v FPGA

V aplikačně specifických architektur se typicky předpokládá běh předem známé aplikace. Proto je možno též používat takové výpočetní modely, které souběžnost popisují deterministicky (např. konečný automat, SCP, SDF). Pokud je použití nedeterministického modelu (předpoklad neomezené velikosti paměti – viz Turingův stroj) nezbytné, pak je vhodné zabudovat do systému takové mechanizmy, které umožní nedeterministické chování detekovat a ošetřit viz např. kapitola 8.3.

V oblasti heterogenních systémů je nezbytné provádět jejich specifikaci s využitím různých modelů podle potřeby. Dále je třeba transformovat vytvořenou specifikaci do validní implementace a rozumět dopadům různých výpočetních modelů na dobu návrhu (okamžik uvedení na trh) a implementační možnosti (technologie) a volbu prvků (komponenty) viz další kapitoly.

6 ODHADY A OPTIMALIZACE

Pro optimalizaci rozdělení systému mezi části vykonávané v SW a HW je nezbytná kvantitativní analýza (evaluace a odhady) vlastností jak jednotlivých částí a jejich interakce, tak chování výsledného systému. Kontext problematiky odhadů a optimalizace při platformním HSC návrhu výpočetních systémů je znázorněn na Obr. 27.



Obr. 27 Odhady a optimalizace

Zjišťování vlastností HW implementovaného v obvodech FPGA je, díky pokročilým technikám syntézy, které standardně předávají informaci o ceně (počet použitých komponent a spojů) a výkonnosti obvodu (zpoždění komponent a spojů), na poměrně vysoké úrovni. V případě SW je však často třeba provádět odhady na základě analýzy testovacích programů (rychlost vykonávání, potřebná kapacita paměti). Pro jednoduchost se často předpokládá, že se jedná o sekvenční úlohy (např. SW čeká na výsledek výpočtu v HW) a výkonnost systému se tedy odhaduje na základě znalostí o vykonávání sekvenčně řazených úloh. V případě souběžných úloh zpracovávaných na více výpočetních jednotkách, které využívající hierarchii paměti jsou odhady velmi náročné [72].

Vzhledem ke složitosti analytických odhadů výkonnosti a potřebné kapacity paměti se v případě použití platforem s FPGA jeví jako efektivnější, odhadovat vlastnosti systému použitím tzv. benchmarků. Při použití platforem DSPX, DX6 a DX64 lze pro DSP algoritmy velmi dobře stanovit výkonnost systému využívajícího příslušný DSP procesor, neboť jsou předem známy doby vykonávání a paměťové nároky použitých algoritmů (FIR, FFT atd.). Situace je však složitější v případě, že je výpočet rozložen mezi různé výpočetní prostředky, které spolu komunikují. Zde je třeba zahrnout i vlastnosti příslušného rozhraní a jeho využití jednotlivými úlohami, kdy se často chování systému dá ověřit až po jeho implementaci. Rozdělení úlohy se tedy, do značné míry, řídí na základě kvalifikovanými odhadů. Výhodou platforem s FPGA je, že alokaci, mapování a plánování lze iterativním způsobem optimalizovat tak, že se úloha rozdělí na základě předběžných odhadů, implementuje se, měřením se zjistí skutečné parametry a provedou se případné korekce.

Nízký příkon a úspory energií se stávají v posledních letech stejně důležitým faktorem při HSC návrhu systémů jakými jsou tradičně výkonnost a cena. Podle názoru autora právě optimalizace s ohledem na nízký příkon bude v následujících letech nejdůležitější aspektem návrhu výpočetních systémů. V případě technologie FPGA je možno výpočetní struktury škálovat, což může, ve spojení s GALS návrhovými technikami, usnadnit tvorbu výkonných a přesto energeticky úsporných systémů viz následující odstavce.

6.1 PŘÍKON A ENERGIE

Příkon nelze snižovat izolovaně nýbrž v kontextu ceny, výkonnosti, výtěžnosti výroby, spolehlivosti, integrity signálu atd. Složitější techniky omezení příkonu vedou na složitější návrh, validaci a implementaci systémů. Často je nezbytné kombinovat různé techniky, což opět návrh komplikuje. Důležitá je konzistence použitých přístupů, kdy použité techniky zachovávají přínosy těch ostatních. Dále platí, že pro kvalitní výsledky musí být techniky návrhu pro nízký příkon součástí celého návrhového procesu, od systémové úrovně, až po úroveň logických obvodů.

Hustota tepelného výkonu

Příkon integrovaného obvodu se při jeho činnosti přemění na teplo, které je třeba odvést (vyzářit do prostoru). Produkovaný tepelný výkon koreluje s Mooreovým zákonem v tom, že složitost integrovaných obvodů a hustota tepelného výkonu (vyzářený výkon na plochu čipu) se zdvojnásobuje přibližně každých 18 měsíců [40]. Za technologický limit lze považovat hustotu tepelného výkonu, který je třeba odvést z nukleárního reaktoru (cca 250 $W \cdot cm^{-2}$), kdy dnešní procesory mají hustotu tepelného výkonu, která se blíží hodnotě kolem 100 $W \cdot cm^{-2}$. Pokud by trend v nárůstu frekvence procesorů pokračoval jako doposud (při konstantním napájecím napětí), pak odhady říkají, že tohoto limitu by se dosáhlo kolem roku 2015 [26]. Toto je jedním z důvodů, proč se návrhové metody pro nízký příkon dnes stávají tak aktuální.

Tlak na nízkou cenu zařízení, kde klíčová položka ve skladbě ceny je dána okamžikem uvedení produktu na trh, je dnes hlavním důvodem proč jsou preferovány výpočetní prostředky, které jsou programovatelné (MPU není třeba navrhovat, stačí jen naprogramovat). Na druhou stranu je ale MPU z hlediska příkonu méně efektivní, než třeba obvody ASIC, které mají optimalizovanou strukturu a nabízí proto nejlepší poměr příkon – výpočetní výkon (ASIC však nelze snadno programovat a jeho návrh trvá dlouho). Jak již bylo řečeno, technologie FPGA v sobě spojuje jak možnost programování, tak optimalizace výpočetních struktur. S cenou též souvisí zabezpečení dlouhé životnosti zařízení, kdy platí, že spolehlivosti klesá s provozní teplotou (menší příkon znamená též nižší pracovní teplotu). Omezení příkonu je důležité s ohledem na konstrukci zdrojů, životnost a omezenou kapacitu baterií u přenosných zařízení. V neposlední řadě určuje potřebný příkon zařízení konstrukci zařízení – rozvody napájecího napětí (průřez vodičů), návrh desky s plošnými spoji, chladič atd. A konečně nižší příkon (omezení proudových špiček na rozvodech napájecího napětí) vede na menší rušení a větší odolnost proti šumu (elektromagnetická kompatibilita).

CMOS technologie

U dnes nejpoužívanější technologie CMOS je příkon určen především tzv. dynamickým příkonem [29]:

$$P_{dyn} = C_{eff} \cdot V_{dd}^{\ 2} \cdot f \text{ , kde} \qquad (2.)$$

V_{dd} je napájecí napětí obvodu, C_{eff} je tzv. efektivní parazitní kapacita všech uzlů v obvodu (skutečně buzená přechody z log. 0 do log. 1 a naopak) a f je frekvence, na které obvod pracuje.

Příkon tedy klesá se čtvercem poklesu napájecího napětí V_{dd} ; lze jej snížit též omezením četnosti změn logických úrovní v uzlech obvodu a zmenšením efektivní parazitní kapacity C_{eff} (menší elementární rozměry integrovaného obvodu, např. přechod z 90nm technologie na 45nm).

6.2 METODY SNIŽOVÁNÍ PŘÍKONU

Problematiku snižování příkonu lze řešit řadou způsobů od nastavení fyzikálních parametrů obvodu až po volbu a způsob použití výpočetních prostředků.

Základní metodou pro snížení příkonu je snížení napájecího napětí, které lze provádět i dynamicky. Tato technika je velmi účinná (standardně používá se u některých MPU) a v případě komerčně dostupných FPGA lze dosáhnout úspory příkonu přes 50 % [30].

Příkon lze též omezit použitím doménově orientovaných architektur. Např. u DSP výpočetních jednotek, které jsou optimalizovány pro vykonávání konvoluce (operace násob – akumuluj), je vhodné používat specializované adresovací režimy pro přístup k datům a koeficientům. Jedná se např. o tzv. modulo adresování či bitově reverzní adresování, která omezují počet instrukcí potřebných pro výpočet ukazatele do pole hodnot v paměti. Dále je výhodné použití aritmetických operací se saturací, kdy není třeba ošetřovat přetečení (chybu lze často zanedbat), nebo zaokrouhlování, použití desetinných čísel se znaménkem a aritmetika s pevnou řádovou čárkou (dynamický rozsah je předem znám a neplýtvá se zbytečně energií pro vykonávání složitých operací v plovoucí řádové čárce).

Mezi další techniky patří místní řízení napájecího napětí, kdy některé subsystémy mají jiné napájecí napětí než jiné dle toho, jakou výkonnost mají mít. Tento přístup je aplikován i u systémů sestavených na deskách s plošnými spoji. Je třeba jen zajistit, aby jednotlivé subsystémy mezi sebou komunikovaly pomocí stejných logických úrovní. Díky flexibilitě FPGA, u kterých lze využít prakticky všechny standardy logických úrovní, je jejich integrace s obvody pracujícími na jiném napětí možná.

Další možností je dynamické řízení frekvence, na které obvod pracuje dle potřeb běhu aplikace. Např. při přecházení mezi režimy činnosti, jako jsou „normální činnost“, „čekání na přerušení" (periferie pracují, výpočetní jednotky mají vypnuté hodiny) a „spánek“ (hodiny celého obvodu jsou vypnuty a lze jej vzbudit pomocí externího signálu).

V FPGA lze dynamický příkon omezit též omezením četnosti přechodů v uzlech obvodu. Jedná se např. o vypnutí hodinových rozvodů, které nejsou využity, řízení distribuovaných pamětí tak, aby se omezily změny na jejich výstupech, řízení povolovacích hodinových vstupů CE (angl. Clock Enable) klopných obvodů, atd. Dále lze optimalizovat návrh (např. s použitím interaktivních rozmísťovacích nástrojů jako je Floorplanner firmy Xilinx) s cílem zkrátit délky vodičů (omezení parazitních kapacit) a počtu buzených vstupů. Jedná se např. rozdělení obvodu synchronizovaného jedním hodinovým signálem na více sekcí.

V následujících odstavcích si uveďme některé techniky omezení příkonu, které byly při tvorbě systémů s FPGA platformami aplikovány.

Napájecí napětí a frekvence

Příkon integrovaných obvodů roste lineárně s jeho složitostí (počtem tranzistorů) a s určitou složitostí obvodu může tedy jeho příkon P vzrůst nad mez, nad kterou nebude možno vyzářené teplo odvést. Jak již bylo řečeno, příkon obvodů vyrobených technologií CMOS lze snížit se čtvercem napájecího napětí V_{dd} . Je však třeba dodržet technologická omezení [39], kdy je jednak nezbytné úměrně snížení napětí třeba snížit též pracovní frekvenci f obvodu (pro zpoždění

tranzistorů totiž CMOS platí, že $\tau \approx 1/V_{dd}$ a tedy $f \approx V_{dd}$) a dále se musí dodržet minimální napájecí napětí CMOS technologie (limit je kolem 0,5 V[12]).

Uveďme si příklady, jak dosáhnout omezení příkonu při zachování výkonnosti, případně zvýšení výkonnosti při zachování příkonu: Vezměme např. obvod s jistým příkonem P_1. Předpokládejme, že danou úlohu lze zpracovat paralelně pomocí více výpočetních jednotek se stejnou složitostí. Nyní těmto jednotkám snižme napájecí napětí a tedy i pracovní frekvenci na polovinu. Tyto nové obvody budou tedy mít každý poloviční výkonnost (poloviční frekvence, na které pracují) a jejich příkon bude $P_2 = C_{eff} \cdot (V_{dd}/2)^2 \cdot f/2 = P_1/8$, tedy osmina příkonu původního obvodu. Pokud potřebujme zachovat původního příkon P_1, pak lze paralelním zpracováním pomocí 8 jednotek pracujících na poloviční frekvenci dosáhnout 4× větší výkonnosti (avšak s použitím 8× většího počtu tranzistorů).

Díky snížení pracovní frekvence obvodů se zjednodušuje i implementace, neboť nejsou takové nároky na kvalitu rozmístění a propojení s ohledem na zpoždění propojovací sítě. V neposlední řadě je snížení pracovní frekvence obvodu výhodné i s ohledem na menší rušení (elektromagnetická kompatibilita) a pomalejší obvody jsou typicky i levnější. Nároky na větší potřebnou kapacitu FPGA totiž nemají takový vliv na jejich cenu (díky platnosti Mooreova zákona) jako na rychlost (ve výrobě se obvody třídí podle rychlosti a platí, že výtěžnost rychlejších čipů je nižší než pomalejších). FPGA platformy též (sdružení komponent na jednom čipu), mají nižší energetické nároky díky interní komunikaci (při komunikaci na deskách s plošnými spoji je třeba budit výrazně větší parazitní kapacity vodičů).

V řadě aplikací s FPGA platformami byly výše uvedené techniky úspěšně využity. Jako příklad optimalizace lze uvést úlohu, ve které bylo třeba nalézt takové řešení, při kterém bude mít výpočetní platforma s FPGA (inteligentní kamera viz odstavec 4.8) nižší příkon z důvodů nezbytnosti bateriového napájení (viz též odstavec 4.8). Zároveň však bylo třeba zajistit vyšší výpočetní výkon, přičemž současné řešení pracovalo na hranici limitů stanovených normami pro elektromagnetickou kompatibilitu (konkrétně vyzařování na pracovní frekvenci 100 MHz). V původní platformě byl použit obvod FPGA, který pracuje s napájecím napětím 1,8 V a je vyroben 150nm technologií. Po analýze možností byl jeho náhrada vybrán obvod, který pracuje na napětí 1,2 V a je vyroben 90nm technologií a má tedy i úměrně menší efektivní kapacity C_{eff}. Cena těchto obvodů je prakticky stejná s tím, že náhrada má zhruba dvojnásobnou kapacitu (díky novější technologii výroby). Odhad příkonu optimalizované jednotky byl stanoven jako $P_{opt} = 90/150 \cdot (1{,}2/1{,}8)^2 \cdot 66/100 = 0{,}176 \cdot P$. Ve výsledném obvodu byla daná výpočetní jednotka replikována 3× s tím, že všechny jednotky nyní pracují na 66 MHz. V důsledku toho bylo odhadnuto, že se příkon sníží cca na polovinu ($3 \cdot P_{opt} = 3 \cdot 0{,}176 \cdot P = 0{,}528 \cdot P$). Obvod tedy pracuje téměř s dvojnásobným výpočetním výkonem (výkonnost je úměrná pracovní frekvenci výpočetních jednotek a v sumě tedy dostáváme 3×66 MHz = 198 MHz oproti původním 100 MHz) a bude mít i lepší vlastnosti s ohledem na elektromagnetickou kompatibilitu.

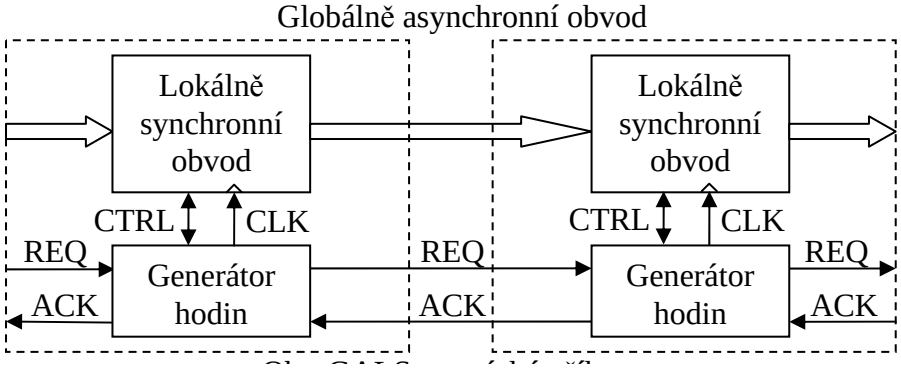
Globálně asynchronní a lokálně synchronní obvody

U hodinově synchronních obvodů implementovaných v FPGA je velká část dynamického příkonu dána generováním a především distribucí hodinového signálu na čipu. Důvodem je, že s ohledem na dnes dominantní hodinově synchronní návrh HW, je třeba vybudit hodinové vstupy všech klopných obvodů pokud možno v jednom okamžiku a na co nejvyšších frekvencích (s požadavkem na co největší výkonnost). Generování a rozvod hodinového signálu vede na

12 Dnešní FPGA (Virtex5 firmy Xilinx) mají napájecí napětí 1 V a mezní frekvence, na které pracují interní výpočetní struktury je 550 MHz [68].

složitý subsystém, který zahrnuje generování hodin, děličky a násobičky kmitočtu. S ohledem na konečnou rychlost šíření signálu je třeba též zajistit synchronizaci hodinových signálů jak v rámci daného čipu, tak mezi obvody na desce s plošnými spoji pomocí fázových závěsů (angl. Phase Locked Loop – PLL) a číslicových zpožďovacích linek (angl. Delay Locked Loop – DLL) atd. Doba šíření hodinového signálu v obvodu je často řádově srovnatelná s periodou hodinového signálu, což vede na velké nároku na zajištění stejné doby šíření ke všem klopným obvodům v systému. Jistého urychlení přenosu informace na čipu lze dosáhnout použitím spojů s lepší vodivostí (měď místo hliníku), větším počtem propojovacích vrstev a trojdimenzionální organizací, vždy se nakonec bude narážet na fyzikální limity (rychlost šíření světla).

S ohledem na potřeby zvyšování výkonnosti je jedním z řešení rozdělit logické obvody na menší subsystémy, které jsou hodinově synchronní lokálně a komunikují mezi sebou asynchronně (GALS obvody). To je v souladu i s potřebami snižování příkonu a specifikací a modelováním systémů (viz odstavec 5.2). Příkon lze pak omezit vypínáním hodinového signálu GALS obvodům [25], které např. čekají na data. Další možností je výrazně snížit frekvenci hodinového signálu. Hodinový signál na plné frekvenci se pro jednotlivé subsystémy zapíná pouze tehdy, pokud je jejich činnost požadována (mají na vstupu data). Každý subsystém může mít své lokální generátory hodinového signálu a vzájemně komunikují asynchronně. V celém systému tedy nemusí být globální rozvod hodinového signálu, což je velmi výhodné i při integraci subsystémů (např. FPGA s DSP) na deskách s plošnými spoji. Na Obr. 28 je principielní schéma GALS obvodu, který podporuje adaptaci frekvence hodinového signálu. Tento obvod využívá komunikačního mechanizmu modelu CSP.



Obr. 28 GALS obvod

GALS techniky byly s úspěchem použity i pro platformy s FPGA, které mají několik nezávislých rozvodů hodinových signálů.

6.3 PŘÍKON A VÝKONNOST

Odhad zrychlení úlohy, kterého lze dosáhnout akcelerací její jisté části, se tradičně demonstruje pomocí tzv. Amdahlův zákona [1]. Tento zákon říká, že pokud se určitá část p dané úlohy urychlí její paralelizací $N\times$, pomocí N výpočetních jednotek (zřetězeně či paralelně), přičemž zbytek úlohy s bude dále zpracován sekvenčně (uvažujme plné vytížení všech N jednotek, zanedbejme vliv paměti a režii), pak urychlení, kterého lze dosáhnout je:

$$S_A = \frac{s+p}{s+\frac{p}{N}}. \tag{3.}$$

V limitním případě, kdy $N \to \infty$, lze tedy dosáhnout zrychlení $S_{A(N\to\infty)} = (s+p)/s$. Např. pokud je možno urychlit 90 % úlohy, bude se celkové zrychlení, kterého je možno teoreticky dosáhnout, blížit k hodnotě $S_{A(N\to\infty)} = (0{,}1+0{,}9)/0{,}1 = 10$. Při použití např. 10 jednotek bude zrychlení $S_{A(N=10)} = (0{,}1+0{,}9)/(0{,}1+0{,}9/10) = 5{,}26$.

V praxi je situace, kdy máme takovou úlohu, ve které je sekvenční a paralelizovatelná část neměnná, méně častá. Vyššího stupně urychlení se dosahuje tím, že zvýšíme množství práce, které bude zpracovávat paralelizovaná část úlohy. Např. budeme zpracovávat větší objemy dat, nebo dělat novou práci, kterou nebylo možno provádět v případě, kdy se úloha neurychlovala. Tento skutečnost popisuje tzv. Gustafsonův zákon [24], který říká, že celková práce, jež systém zpracuje při použití N akcelerátorů, je rovna:

$$S_G = s + N \cdot p\,. \tag{4.}$$

Dále, pokud předpokládáme, že sekvenční část úlohy již byla urychlena jejím zřetězeným zpracováním pomocí M stupňů, lze psát:

$$S_{GZ} = \frac{\frac{s}{M} + p \cdot N}{\frac{s}{M} + p}\,. \tag{5.}$$

Problém ale nastává, pokud musíme zohlednit příkon akcelerovaného systému. Předpokládejme, že potřebujeme zrychlit vykonávání dané úlohy, ale musíme zachovat příkon systému. Příkon je možno omezit snížením napájecího napětí a pracovní frekvence výpočetních jednotek. Např. v případě akcelerace úlohy pomocí 10 jednotek, bude třeba snížit jak jejich napájecí napětí, tak pracovní frekvenci na hodnoty, při kterých budu akcelerátory pracovat s 10× nižším příkonem $P_{1/10}$. Jelikož C_{eff} je konstanta (předpokládáme stejné jednotky vyrobené stejnou technologií) a pro normalizovanou frekvenci a napětí lze psát $f \approx V_{dd}$, pak je příkon jedné akcelerující jednotky roven $P_{1/10} = P/10 = (C_{eff} \cdot {V_{dd}}^2 \cdot f)/10 \approx {V_{dd}}^2 \cdot f/10 \approx f^3/10$. Pro příkon akcelerátorů platí $P_{1/10} \approx {f_{1/10}}^3$ a tedy ${f_{1/10}}^3 = f^3/10$. Jelikož jsme předpokládali normalizovanou frekvenci původního obvodu, pak normalizovaná frekvence akcelerátorů je $f_{1/10} = \sqrt[3]{1/10} = 0{,}47 \cdot f$. Neboť výkonnost akcelerátorů je dána jejich pracovní frekvencí, lze pro výsledný obvod říci, že urychlení úlohy 10 akcelerátory bude rovno 47 % hodnoty, které měla úloha bez omezení příkonu.

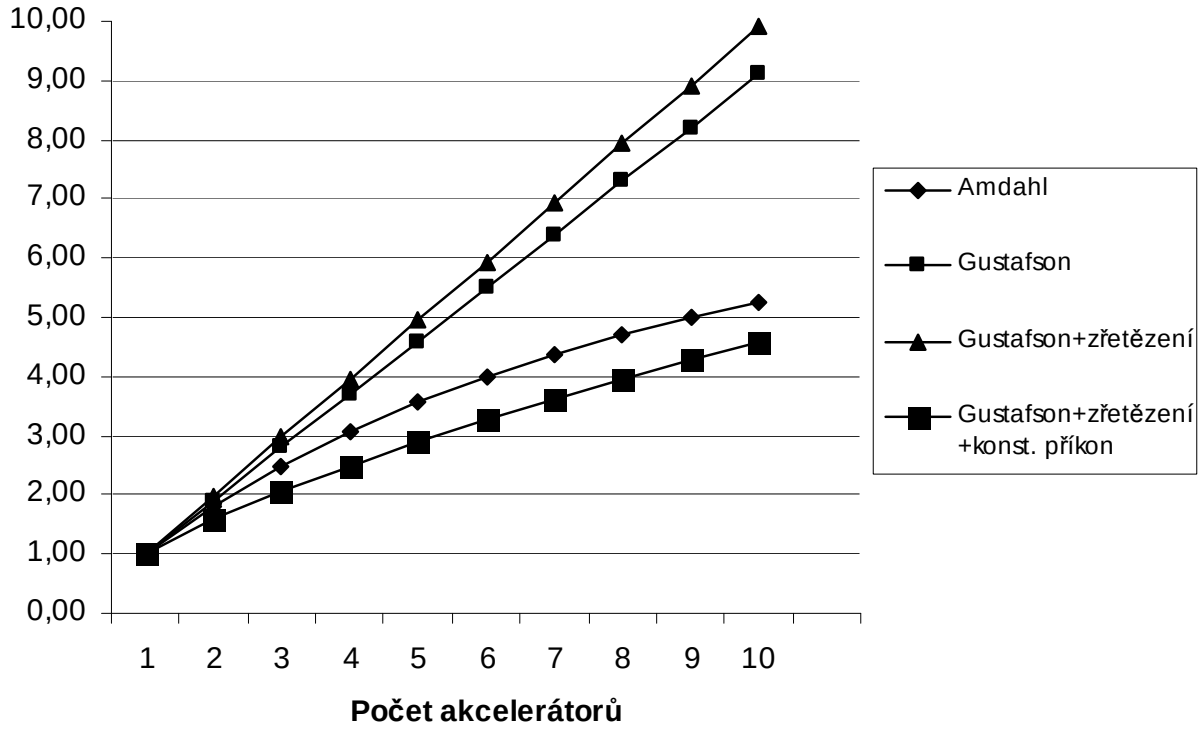
Při konstantním příkonu lze tedy zrychlení úlohy v závislosti na počtu akcelerátorů odhadovat vztahem:

$$S_{Pconst} = \frac{S}{\sqrt[3]{N}}\,. \tag{6.}$$

Např. v případě odhadu provedeného pomocí Gustafsonova zákona, za předpokladu zřetězení sekvenční části výpočtu $M\times$, použití N akcelerátorů a při zachování příkonu, bude zrychlení dáno vztahem:

$$S_{GZPconst} = \frac{\frac{s}{M} + p \cdot N}{\frac{s}{M} + p} \cdot \frac{1}{\sqrt[3]{N}}. \qquad (7.)$$

Na Obr. 29 uveden graf zrychlení 90 % úlohy v závislosti na počtu akcelerátorů pro výše uvedené případy. V případě zrychlení $S_{GZPconst}$ pro jednoduchost předpokládejme, že počet stupňů zřetězení je stejný jako počet akcelerátorů ($M = N$) Maximální zrychlení pro deset akcelerátorů a stupeň zřetězení 10 je pak roven $S_{GZPconst(N=M=10)} = 4{,}65$.



Obr. 29 Zrychlení úlohy

Uvedené odhady lze použít při návrhu aplikačně specifických úloh, u kterých lze většinou předem naplánovat jak využití zřetězených, tak paralelních výpočetních jednotek. U univerzálních procesorů je však situace poněkud odlišná. Pollackovo pravidlo říká, že výkonnost mikroarchitektur univerzálních procesorů roste přibližně se čtvercem jejich složitosti (počet tranzistorů). Např. pro 2× větší výkonnost procesoru je třeba až 4× více tranzistorů (které též mají 4× většího příkon). Toto pravidlo odhady parametrů FPGA platforem využívajících hard IPC procesory komplikuje, protože skutečně odvedená práce neroste úměrně s rostoucí složitosti systému. Pokud je režie aplikačně specifická a lze ji odhadovat např. pomocí benchmarků. Při univerzálních výpočtech se obtížně odhaduje režie spojená s přístupy do paměti, synchronizací úloh, stupněm využití zřetězených jednotek atd. Může zde tedy nastat i situace, kdy při použití 10× složitějšího obvodu (a neomezeném příkonu), dostaneme zrychlení jen kolem $S_{N=10} \approx \sqrt{10} = 3{,}16$.

6.4 SHRNUTI

Při modelování a implementaci úloh je účelné, s ohledem na charakter aplikace, definovat, zda je hlavní motivací zvýšení propustnosti (zpracovaný objem dat za jednotku času), nebo je cílem

zkrácení latence (reaktivní systémy), kde je hlavním parametrem rychlost odezvy na vnější (typicky asynchronní) podněty. Výpočetní systémy implementované pomocí FPGA platforem umožňují optimalizaci jak s ohledem na propustnost (rozdělení úlohy na více paralelních jednotek), tak latenci (více paralelních jednotek, pro každou úlohu jedna). Jejich návrh je zjednodušen tím, že jsou dobře škálovatelné. Hlavní přínos z hlediska výkonnosti je možnost využití distribuované hierarchie paměti a hybridních výpočetních struktur (viz kapitola 3).

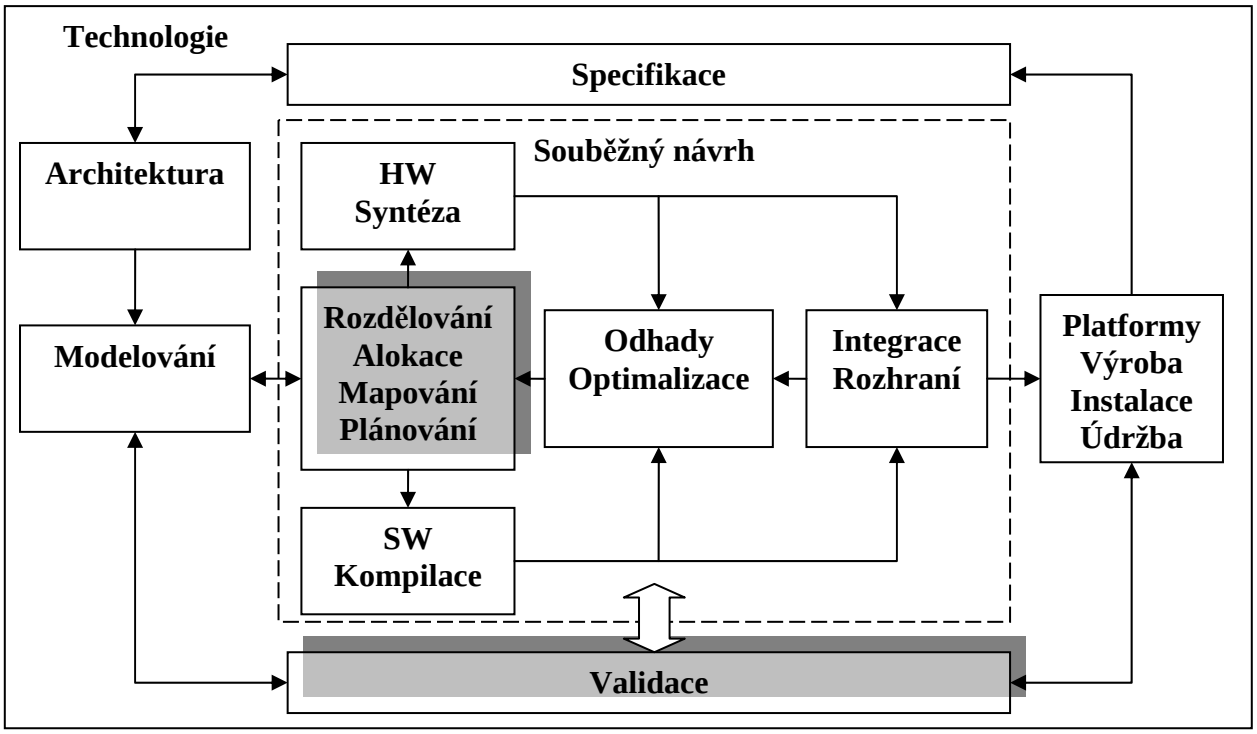
Tradičním způsobem urychlování běhu aplikací je zrychlením těch jejich částí, jichž výpočet trvá nejdelší dobu. Např. u transformačních úloh (číslicové zpracování signálů) trvá nejvíce času vykonávání poměrně jednoduchých částí algoritmů, které se však velmi často opakují. Typicky se jedná o programové smyčky, jejichž jádro může být implementovatelné v HW s využitím poměrně malého počtu výpočetních struktur. Pro analýzu úloh a nelezení těchto částí se používá řada přístupů. Tradičně se využívá profilování, kdy se úloha původně napsaná v SW spustí a na základě statistiky četnosti vykonávání jednotlivých částí úlohy a doby v nich strávené, se izolují časově kritické části. Další možností je analytický přístup, kdy se odhadují pravděpodobnosti vykonání jednotlivých částí úlohy. Informativně lze pak např. pomocí Amdahlova zákona stanovit, nakolik se urychlení izolované části algoritmu projeví v celkovém urychlení běhu celé aplikace a jestli se tedy urychlování vyplatí. Pokud je zrychlení dotaženo přidáním výkonnější výpočetní jednotky, než je bezprostředně třeba pro urychlení dané části úlohy, lze do ní přesunout i jiné části výpočtu či dokonce implementovat nové funkce. Výsledné zrychlení lze pak odhadnout pomocí Gustafsonova zákona. Často používaným přístupem k řešení tohoto problému je lokalizace vhodných částí algoritmů, jejichž implementace v HW je možná s maximální mírou paralelizmu. Výpočet zbytku aplikace se pak provádí sekvenčně. Je zde třeba vyřešit mechanizmus synchronizace výpočtů běžících v paralelně (v HW – např. FPGA) a sekvenčně (SW v MPU), jakož i předávání dat. Další často používanou metodou je využití výjimek. Příkladem je např. HW jednotka (např. subsystém směrovače paketů), která analyzuje a zpracovává běžný tok na síti (často se opakující pakety) a pokud zjistí, že daný paket je nějakým způsobem nestandardní, předá jeho zpracování do MPU. Tento koncept je např. využit na platformě Liberouter viz odstavec 4.7. Pokud je třeba navrhovat systémy při zachování konstantního příkonu, lze zrychlení odhadovat např. pomocí vztahu (6).

Ve výpočetních systémech pro všeobecné použití (např. PC s operačním systémem) je využití paralelizmu obtížné. Dnes dominantní technikou, jak souběžnost zavést je rozdělení programu na vlákna. Procesory podporují vykovávání vláken na úrovni mikroarchitektury. Ovšem počet paralelně vykonávaných vláken je omezen složitostí HW, který ji řídí. Důvodem je potřeba vykonávání obecně libovolného programu a není tedy možno předem optimalizovat jeho vykonávání pro konkrétní procesor, operační systém a kontext (přepínání procesů). Důvodem pro potřebu sofistikované HW podpory vícevláknového vykonávání je skutečnost, že uvedené postupy se aplikují nad standardními soubory instrukcí, kdy je třeba dodržet zpětnou kompatibilitu kódu. Proto je použití MPU v FPGA pečlivě vážit a používat jen tehdy, pokud se to vyplatí s ohledem na rychlost návrhu.

Významného urychlování výpočtů úloh lze, při dodržení nízkého příkonu, dosáhnout kombinací univerzálních výpočetních jednotek a specializovaných výpočetních jednotek, které využívají dekomponovanou hierarchii paměti. Tento trend je patrný jak v oblasti návrhu SoC, tak při aplikacích s obvody FPGA.

7 ROZDĚLOVÁNÍ

Rozdělení systému na menší celky umožňuje zefektivnit jak jeho modelování (díky možnosti aplikovat nejvhodnější model pro každý subsystém viz kapitola 5), tak dosáhnout i efektivnější implementace (viz např. GALS techniky uvedené v kapitole 6). Kontext problematiky rozdělování při platformním HSC návrhu výpočetních systémů je znázorněn na Obr. 30.



Obr. 30 Rozdělování: alokace, mapování, plánování

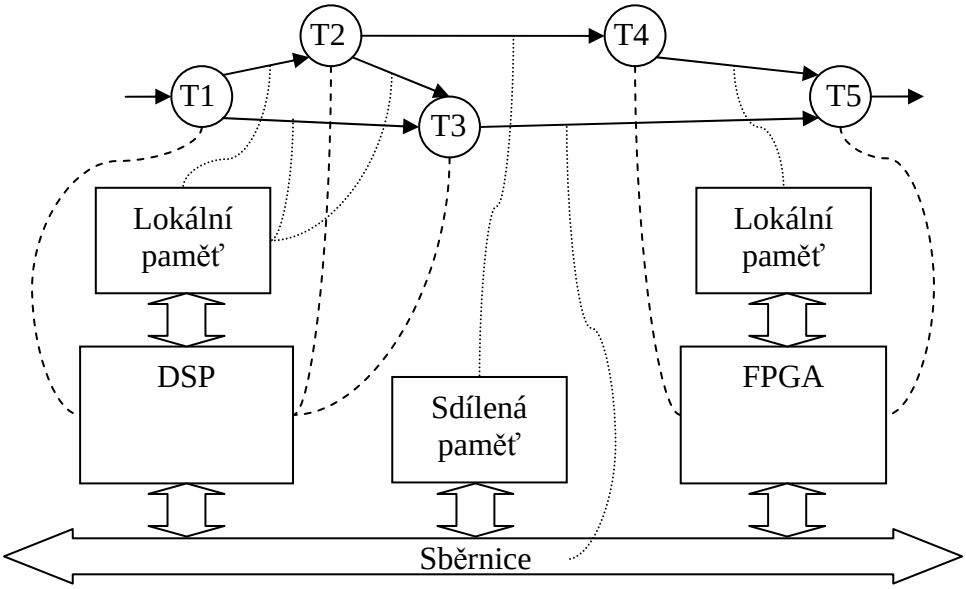
V kontextu rozdělování výpočetních systémů byly publikovány prakticky všechny klasické optimalizační algoritmy, viz např. přehled v [13]. Většinou se jednalo o optimalizaci HSC problémů s malou zrnitostí komponent kde vzhledem k velikosti prostoru možných řešení, není nemožné jeho expertní prohledávání. Někteří autoři kombinují heuristické přístupy pro prvotní rozdělení a následně jej optimalizují pomocí exaktních algoritmů [66]. Problematice rozdělování se též podrobněji věnovat autor ve své disertační práci [20], ve které zformuloval problém rozdělování jako optimalizaci využití logických členů, které se účastní výpočtů. Vzhledem k tomu je zde uveden pouze nástin dané problematiky.

Rozdělování je třeba řešit při návrhu jakéhokoliv systému, který bude ve výrobě sestaven z více, často heterogenních subsystémů viz kapitola 3 . S rostoucím počtem komponent však roste exponenciálně velikost prostoru možných řešení a nalezení rozdělení, které splňuje daná omezení (cena, atd.) je vícedimenzionální optimalizační úloha, z nichž většina je NP-úplných. Rozdělování a integrace jsou duálními problémy. Dobré rozdělení vede ke snadné integraci systému z jednotlivých subsystémů. Proces rozdělování se skládá z několika dílčích činností. Jedná se o alokaci dostupných výpočetních zdrojů, mapování výpočetních úloh na tyto zdroje a případné plánování jejich využití v čase, pokud se výpočetní prostředky sdílí. K tomu je třeba kvalifikovaně odhadnout, jaké vlastnosti bude výsledný systém mít (viz kapitola 6). Rozdělování je úzce svázáno se specifikací a modelováním výpočetních úloh, neboť často dekompozice aplikace na jednotlivé výpočetní úlohy a zvolený způsob jejich modelování, implikuje použití vhodného výpočetního prostředku a naopak viz kapitola 5 .

Tradiční metody rozdělení úloh mezi SW a HW jsou dvojí: buď je na začátku provedena implementace aplikace v SW a následně se hledají úzká místa (doba výpočtu ap.), která jsou vhodná pro akceleraci v HW, případně se postupuje naopak, kdy se v HW řešení hledají místa, která lze přesunout do SW s cílem zlevnit výsledný systém. Přesuny mezi implementací v SW či HW jsou prováděny buď ručně, nebo vhodnými algoritmy. Optimalizuje se vhodně formulovaná kriteriální funkce, kterou je např. výkonnost u přesunů $SW \rightarrow HW$ či cena u metody $HW \rightarrow SW$. Další možností je, že alokace není předem specifikována ani v SW ani jako HW, ale hledá se takové rozdělení, které využívá vlastností modelů použitých pro modelování a dostupných komponent s cílem nalézt řešení, který vyhovuje všem kritériím.

7.1 ALOKACE, MAPOVÁNÍ A PLÁNOVÁNÍ

Při procesu rozdělování je především třeba stanovit postup alokace komponent, který lze provádět buď shora-dolů či zdola-nahoru. Při postupu zdola-nahoru jsou komponentám přiděleny cílové začátku procesu návrhu (modifikace platformy či znovupoužití IPC). Při postupu shora-dolů se přidělení komponent provádí později s tím, že pozdější rozhodnutí může vést k lepším výsledkům, neboť je k dispozici více času na případná vylepšení či použití novějších technologií. Volba postupu je též dána technologií, která bude použita pro implementaci komponent, kdy např. platformy s FPGA umožňují pozdější modifikace a je tedy možno je zvolit na začátku návrhu. Dalším krokem procesu rozdělování je mapování úloh na alokované výpočetní prostředky. Pokud se nějaký prostředek sdílí více úlohami, je třeba též naplánovat jeho využití v čase. Uvedené činnosti jsou ilustrovány na Obr. 31, kde je uveden příklad přidělení úloh (T1 – T5) na alokované výpočetní a komunikační prostředky.



Obr. 31 Příklad alokace, mapování a plánování

Úlohy T1, T2 a T3 budou vykonávány na DSP procesoru a mezi sebou budou komunikovat přes lokální paměť. U těchto úloh je plán jejich výpočtu dán datovými závislostmi (T1 – T2 – T3). Úlohy T4 a T5 budou zpracovány v FPGA a mezi sebou budou komunikovat přes lokální paměť. Úlohy T2 a T4 budou komunikovat přes sdílenou paměť a sběrnici a úlohy T3 a T5 komunikují po sběrnici.

7.2 ZRNITOST

Kvalitu rozdělování a rychlost návrhu též určuje zrnitost úloh a výpočetních prostředků. Čím menší je zrnitost, tím lepších výsledků lze dosáhnout, neboť prostor možných řešení je větší. Jeho prohledání však trvá delší dobu. Díky stupni integrace dnešních integrovaných obvodů a možnosti použití platforem a IPC se ukazuje, že s ohledem na produktivitu práce a dobu návrhu je často lepší nechat část výpočetních zdrojů nevyužitých či systém předimenzovat. Neméně důležitým parametrem, jako je počet použitých komponent (tranzistorů) a výpočetní výkon, se dnes stává příkon. V případě FPGA platforem se ukazuje, že všechny uvedené veličiny lze vylepšovat škálováním. Dnes lze pozorovat posun při optimalizaci rozdělení od malé zrnitosti spíše k hybridním výpočetním architekturám a výpočetním modelů, které mají velkou zrnitost.

7.3 ODHADY

Rozdělení systému je ovlivněno řadou aspektů, které určují kvalitu výsledku. Parametry, jaké bude výsledný systém mít, však nejsou předem známy (pokud by byly, znamenalo by to, že je aplikace již plně implementovaná) a je tedy nezbytné provést kvantitativní evaluaci dopadů rozdělování na výslednou implementaci. Typicky jsou to funkční omezení jako cena, výkon a příkon a nefunkční omezení, mezi která patří doba uvedení na trh, vzhled, ergonomie atd. Dále jsou to parametry použitých komponent a vlastnosti použitých modelů chování. Např. alokace výpočetního prostředku je provedena např. s ohledem na příkon výsledného systému. Mapování úlohy na výpočetní prostředek je dáno velikostí paměti, kterou má k dispozici atd. Rozdělení systému na menší ceky s sebou přináší nezbytnost komunikace mezi jednotlivými subsystémy a synchronizaci jejich činnosti. Komunikace mezi výpočetními prostředky probíhá přes vhodné médium (sběrnice, sdílená paměť atd.), které je připojeno pomocí daného rozhraní.

Kvalita odhadů určuje kvalitu rozdělení. Ovšem čím je odhad přesnější, tím však déle trvá jeho provedení. Příkladem je tzv. profilování SW, kdy se např. u aplikace popsané v jazyce C sbírají statistiky o jejím chování za běhu (např. počet volání nějaké funkce). Cílem může být např. nalezení časově kritických částí algoritmů, která mohou být vhodnými kandidáty pro akceleraci v HW. Uvedený příklad představuje velmi přesný odhad chování systému, avšak jeho získání vyžaduje jeho faktickou implementaci. Profilování je tedy možné udělat až na funkčním systému či jeho virtuálním prototypu. Častěji ale ani jedno, ani druhé není k dispozici a je tedy třeba provádět odhady jinými metodami. Jedná se např. o analytické odhady četnosti vykonávání určitých částí programu, nalezení nejdelší větve v programu, počet řídících cyklů konečného automatu nezbytných pro vykonání určitého algoritmu v HW, stupeň využití hodinového signálu synchronních obvodů, průměrná doba cyklu na instrukci pro daný program atd. Pro potřeby optimalizace procesu rozdělování při HSC návrhu je třeba též předem kvalifikovaně odhadnout příkon, který bude každý subsystém mít. Odhady příkonu se provádí podobně jako statická analýza časování HW, kdy nás zajímá nejhorší možný a typický příkon. Přesnost odhadů závisí na vstupních podmínkách (frekvence, napětí) a na charakteristických hodnotách specifických pro konkrétní obvod. Např. většina nástrojů pro návrh FPGA umožňuje odhady příkonu (viz např. XPower Estimator firmy Xilinx [www.xilinx.com]).

Rychlost a kvalita, s jakou jsme schopni parametry systému předem odhadnout, určuje počet iterací optimalizace procesu rozdělování, které je možno provést.

7.4 KRITERIÁLNÍ FUNKCE

Na systémové úrovni je, s ohledem na složitost, nezbytné aplikace rozdělit do subsystémů a ty modelovat vhodnými výpočetními modely viz kapitola 5 . Tím je též, do značné míry definováno, jak bude výsledný systém dekomponován. Podobně, v případě použití platforem jsou proto jak výpočetní prostředky, tak do značné míry i aplikační doména dány předem. Rozdělování na

systémové úrovni a při použití platforem lze tedy často provést expertně (člověkem). Pracujeme zde totiž s výpočetními elementy velké zrnitosti a člověk je často schopen provést rozdělení na základě znalostí, zkušeností a intuice. Díky použití heterogenních výpočetních prostředků je často obtížné formálně definovat kriteriální funkci, která by zohledňovala jejich vlastnosti dostatečně přesně. Návrh složitých systémů je též týmová práce, kde se s výhodou dá využít znalostí a zkušeností více lidí. V praxi se velmi se osvědčilo jak při specifikaci, tak rozdělování systému využití technik tzv. brainstormingu, kdy se postupně generují náměty řešení, které se následně evaluují. Často lze tímto přístupem dospět k inovativnímu řešení, kterých by čistě formálními postupy nebylo možno nalézt.

S ohledem na velkou zrnitost řady úloh může být tedy často rychlejší provést rozdělení expertem, než formulovat vhodnou kriteriální funkci, která pak určuje kvalitu rozdělení. Kriteriální funkci navrhuje člověk na základě expertních znalostí dané problematiky. Pokud ji vytvoří dobře, může nějaký optimalizační algoritmus nalézt dobré rozdělení. Používání automatizovaných postupů se pak vyplatí, jen pokud je třeba vytvářet řadu podobných aplikací. Pokud se podaří s danou kriteriální funkcí nalézt vyhovující rozdělení, může se tato následně upravovat podle nových požadavků a rutinně používat pro hledání rozdělení nových problémů. Tento postup lze s úspěchem použít např. při návrhu jednodušších systémů (např. biologií inspirovaný návrh HW [59]).

Kriteriální funkce typicky zahrnuje řadu parametrů, jako je cena, výkonnost, příkon apod. Je však obtížné vytvořit ji tak, aby též zahrnovala parametry rozhraní mezi subsystémy v závislosti na rozdělení systému, kdy přesun funkčnosti může implikovat změnu vlastností rozhraní. Dále platí, že návrh systémů není vždy motivován výkonností či cenou řešení, ale závisí též na „iracionálních parametrech“ jako je prodejnost, budoucí rozšiřitelnost apod., které lze definovat jen velmi obtížně.

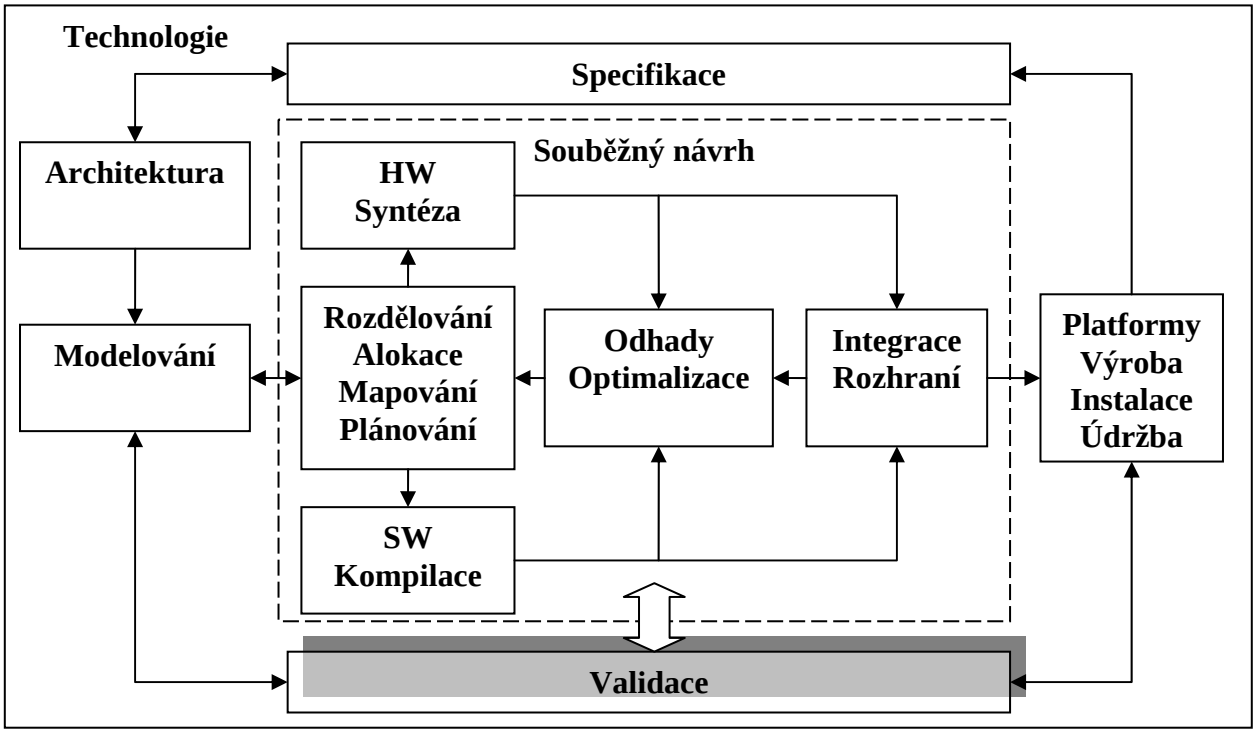
7.5 SHRNUTÍ

Výsledky prohledávání prostoru možných řešení závisí především na složitosti problému a kriteriální funkci, která aproximuje vlastnosti systému. Čím přesnější je aproximace, tím kvalitnější řešení lze nalézt, avšak za cenu větší časové náročnosti. S kriteriální funkcí souvisí i heuristika použitá pro prohledávání prostoru možných řešení. Heuristika je též typicky navržena člověkem a tím je též do značné míry předurčena kvalita nalezeného řešení. Volba heuristiky a tvorba kriteriální funkce musí být provedena expertem se značnými praktickými zkušenostmi v dané oblasti. Algoritmus jen umožní prohledat větší prostor možných řešení a tím zvýšit pravděpodobnost, že bude nalezeno takové, které splňuje dané požadavky. Protože pro všechny heuristické algoritmy v obecném případě platí, že nezaručují nalezení optimálního řešení v dosažitelném čase a jsou závislé na nastavení řady parametrů, které jsou navíc problémově orientované, asi nikdy nebude možné tvrdit, že jeden heuristický algoritmus je univerzálně lepší než jiný. Někteří autoři poukazují na to, že dobrých výsledků se dosahuje spíše použitím efektivnějších heuristik, než lepšími hledacími algoritmy [35].

Rozdělení systému je úloha, ve které jsou jednotlivá kritéria často protichůdná (cena, výkonnost, příkon) a obecně nelze vylepšit jeden parametr, aniž by se nepříznivě neprojevilo na jiném parametru. Pokud se problém rozdělení formuluje např. jako vážený součet jednotlivých kritérií a jejich optimalizace může vést na řešení, jež nezohledňuje některé kombinace daných podmínek. Při návrhu komplexních systémů je tedy nutno optimalizovat dle více kritérií a většinou nelze předem říci, které je nejdůležitější. Často se též mohou podmínky, ve kterých se návrh provádí měnit (změna požadavků uživatele, dostupnost nových technologií), což vede na změnu váhy jednotlivých kritérií. Z tohoto pohledu je výhodné mít více akceptovatelných řešení, ze kterých lze při implantaci vybírat na základě aktuální situace.

8 VALIDACE

Validace je proces, při kterém se zjišťuje, zda produkt či krok při návrhu odpovídá svému účelu, splňuje veškerá daná omezení (cena, výkonnost, příkon ap.) a bude pracovat, jak je vyžadováno. Výsledkem validace by měla být jistota, že byl implementován správný produkt. Jedná se o pohled na produkt z pohledu uživatele. Validace systému se provádí řadou způsobů ve všech fázích návrhu, viz Obr. 32.



Obr. 32 Validace

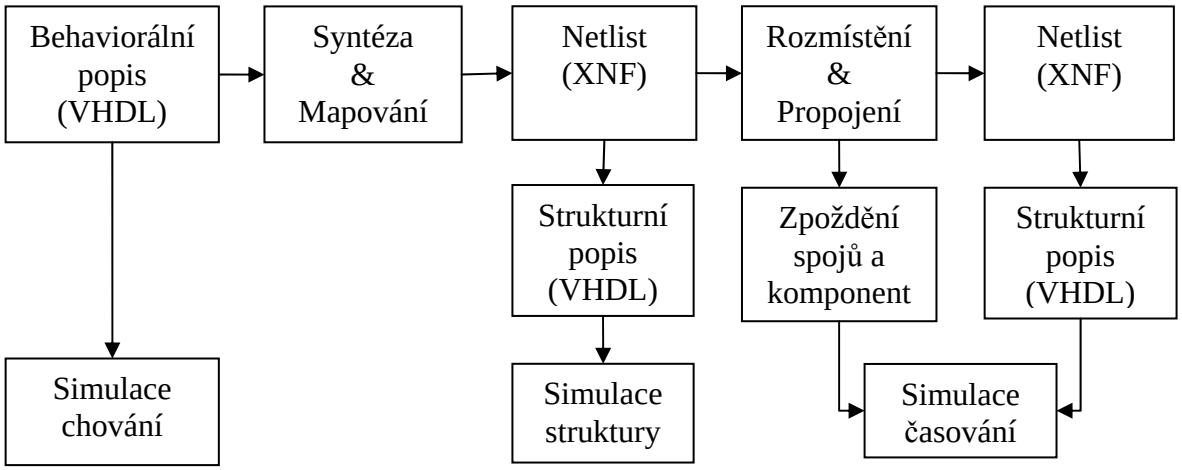
Mezi metody validace patří verifikace, kterou se ověřují dokazatelné vlastnosti (po transformaci modelu z jedné reprezentace do druhé), simulace, která umožňuje kontrolu chování modelu na počítači a testování, které kontroluje chování systémy v reálných podmínkách. Dalšími metodami je emulace, což je fyzické vykonáváním modelu systému na nějakém technickém prostředku a tzv. rychlé prototypování, kdy se vytvoří fyzický model (např. v FPGA se implementuje model procesoru) systému, na kterém lze měřit některé jeho vlastnosti. Konečně za validaci lze považovat i postup, kdy je správnost návrhu zajištěna tzv. konstrukcí (používají se komponenty, které jsou předem ověřené a lze je tedy považovat za správné).

Validace a návrh systému jsou spolu úzce svázány a nelze na ně pohlížet jako na oddělené aktivity. Validace by měla být prováděna již v rámci, či bezprostředně po ukončení jednotlivých etap návrhu produktu tak, aby bylo dosaženo maximálního efektu a jakékoliv neshody by mohly být odstraněny rychle a levně. Validaci chování reálného systému lze provést až na jeho prototypu a někdy až na plně funkčním systému v provozu.

Praktické zkušenosti ukazují, že je třeba kombinovat různé techniky. Nejčastěji se využívají nástroje a postupy, které do značné míry závisí na lidském (subjektivním) rozhodování. Takové techniky jsou dnes nejrozšířenější, neboť pokud jsou aplikovány správně, s použitím formálně definovaných postupů, jsou velmi efektivní.

8.1 SIMULACE

Simulace je tradiční technikou pro validaci návrhu HW. Úroveň abstrakce modelu, na které je vykonáván, je vždy kompromisem mezi dobou simulace a její přesností. Např. v případě návrhu systémů s použitím FPGA (např. od firmy Xilinx) se simulace provádí na několika úrovních, viz Obr. 33.

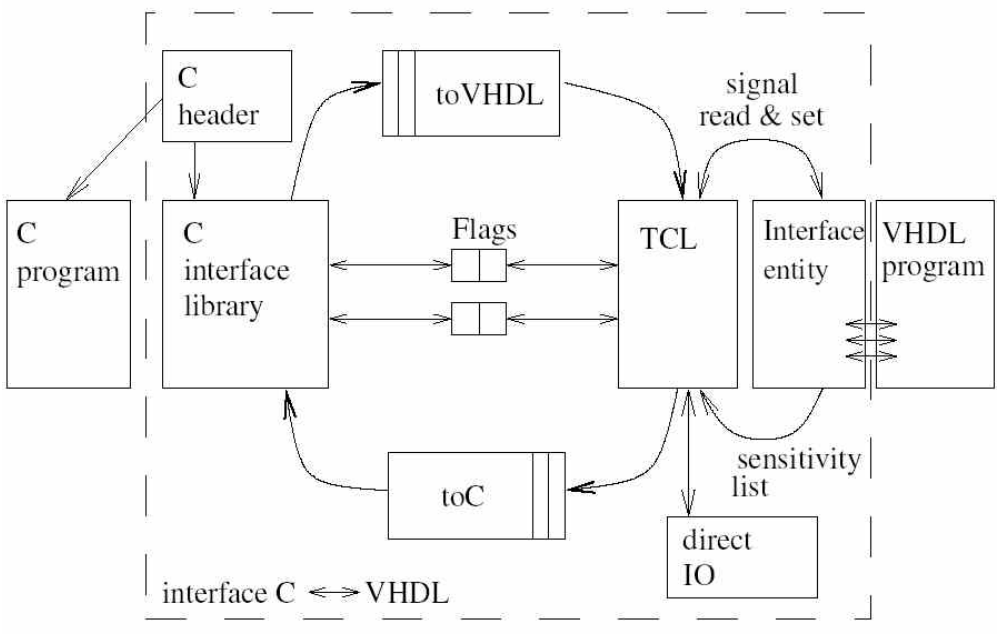


Obr. 33: Funkční a časová simulace FPGA

V prvním případě slouží model ke specifikaci problému a simulací jeho chování se validuje, že návrhář pochopil zadání a vytvořil správný popis. Po syntéze se simulací verifikuje, že byla správně provedena alokace komponent cílové technologie a daný popis na ně byl správně mapován. Struktura obvodu je tedy definována a jeho model je typicky hodinově synchronní (díky vloženým klopným obvodům). V této fázi lze zjistit zpoždění jednotlivých komponent a provést simulaci časování obvodu (bez zpoždění propojovací sítě). Po rozmístění komponent a jejich propojení na čipu je třeba validovat, zda jsou dodržena časová omezení (latence a propustnost) a ostatní kritéria (využitá plocha na čipu, příkon apod.). Uvedené informace nejsou součástí modelu (VHDL kódu), ale je k dispozici jako doplňková informace, kterou generuje příslušný návrhový systém.

SW systému se nejčastěji validuje profilováním, kdy se jednak ověří správná funkce a lze též zjistit jeho chování s ohledem na časování a potřebnou kapacitu paměti.

Podstatou HSC technik je provádění návrhu HW a vývoje SW souběžně, což platí i o simulaci. Příkladem cosimulačního prostředí pro tvorbu HW a SW FPGA platforem je systém [10]. Jedná se o propojení vývojových prostředí pro návrh SW (jazyk C++) a HW (jazyk VHDL) pomocí skriptovacího jazyka TCL. Jazyk TCL podporuje spouštění externích programů a použití rour (pipe) pro směrování vstupů a výstupů. TCL plní rouru událostmi (změna signálu), které jsou generovanými při simulaci HW (simulátor jazyka VHDL) a odesílá je do SW. Podobně jsou odebírány události generované ze strany SW a TCK zabezpečuje jejich odesílání do VHDL simulátoru. Komunikace mazi HW a SW částmi tedy probíhá pomocí zasílání zpráv přes komunikační kanál, kdy použitím roury je zajištěno jejich uspořádání. Pro vzájemné potvrzování se využívá rozšíření jazyka TCK podporované použitým simulátorem jazyka VHDL (ModelSim), které umožňuje pozastavit provádění VHDL procesu, dokud SW část nepotvrdí přijetí události tím, že zapíše do dané proměnné (vwait). Pomocí tohoto poměrně jednoduchého mechanizmu byla simulována vzájemná synchronizace výpočtů prováděných v SW a HW u řady HSC úloh. Na Obr. 34 je principielní schéma komunikace mezi simulátorem HW a prostředím pro vývoj SW.



Obr. 34 Cosimulační prostředí

8.2 VERIFIKACE

Verifikací se obecně ověřuje, zda je model, po transformaci z jedné formy do druhé, ekvivalentní s původním. Formální verifikační techniky jsou založeny na matematických důkazech správnosti modelu. V případě, že lze model formálně verifikovat je toto nejefektivnější způsob, neboť můžeme mít 100% jistotu (avšak pouze za předpokladu, že člověkem vytvořené důkazy jsou správné), že model je správně vytvořen (např. jeho transformací z vyšší úrovně do nižší). Na druhou stranu tvorba formálního důkazu je mentálně velmi náročná (vyžaduje zkušeného experta) a je prakticky omezena jen na relativně jednoduché úlohy. Kvalita verifikace je dána kvalitou použitých důkazů. V případě automatizovaného dokazování např. pomocí tzv. model checkingu [7], může být též problém výpočtem (doba trvání a potřebná kapacita paměti), neboť při kontrole je třeba projít všechny možné kombinace stavů, ve kterých se může modelu nacházet (exponenciální nárůst počtu stavů) a jeho použití je tedy omezeno na relativně jednoduché obvody. Např. v rámci výzkumného projektu Liberouter byly členy týmu aplikovány formální verifikační techniky pro ověření některých vlastností, před jejich implementací [2]. Je však třeba konstatovat, že složitost verifikovaných obvodů byla velmi malá. Lze říci, že formální metody je vhodné používat pro ověření jednodušších stavebních bloků, ze kterých se pak staví složitější obvody.

Formální verifikace návrhu je nezbytná pro eliminaci nevratných chyb s velkými ekonomickými či bezpečnostními dopady (např. při návrhu ASIC, MPU, vestavěných systémů dopravních prostředků). V praxi je při použití FPGA platforem v drtivé většině případů efektivnější provádět validaci systému přímo v reálném prostředí. Důvodem je i to, že validace systému za provozu se stejně musí provést s ohledem na fyzikální vlastnosti (časování, příkon, elektromagnetickou kompatibilitu), které nelze většinou bez výroby prototypu verifikovat (a to ani formálně).

Pro ověření vlastností systémů s platformami se osvědčila aplikace technik pro verifikaci tvrzení (angl. asserts). Popis vlastností se provádí pomocí jazyků pro specifikaci tvrzení, pomocí kterých lze definovat temporální vlastnosti, které má obvod mít a následně je kontrolovat. Jedná se např. o jazyk SystemVerilog [www.systemverilog.org], který rozšiřuje jazyk Verilog (určený pro popis chování, struktury a simulaci) o nové vlastnosti pro usnadnění validace obvodů (popis vlastností a formální verifikaci). Tento jazyk lze též použít pro generování náhodných testovacích

vektorů, čímž lze usnadnit dosažení vysokého stupně pokrytí obvodu testem. Do popisu obvodu se též umísťují příkazy, které umožňují ověřování temporálních závislostí, počítání histogramů hodnot signálů, událostí atd. Uvedené techniky zvyšují pravděpodobnost, že obvod, který byl dostatečně pokryt testem, bude fungovat správně.

8.3 VALIDACE A PLATFORMY

Verifikace a validace je jednou z největších výzev HSC přístupu k návrhu výpočetních systémů. V případě FPGA (oproti obvodům ASIC a SoC) je problém jednodušší v tom, že verifikaci lze provést v reálném systému.

Pro stavbu složitých systémů se ukazuje jako vhodný přístup, který využívá předem ověřených komponent (validace konstrukcí). Např. při využití FPGA platforem se osvědčilo použití IP jader, které byly předem plně otestovány a jejich parametry předem známy. Stále však platí, že s každým novým subsystém (např. IP jádra) je třeba validovat funkci rozhraní a jeho interakci s ostatními subsystémy.

Vzhledem k omezené rychlosti simulace se, při použití platforem s FPGA, může v mnoha případech jevit jako efektivnější ověřit funkci systému na jeho prototypu. Simulací se ověří základní vlastnosti, pak se provede jeho implementace, a chování se testuje v reálných podmínkách. Jednak se odhalí i chyby, které často nelze předvídat, ale především je celý postup výrazně rychlejší. Dále se může provést ověření řady dalších vlastností systému, které je obtížné simulovat.

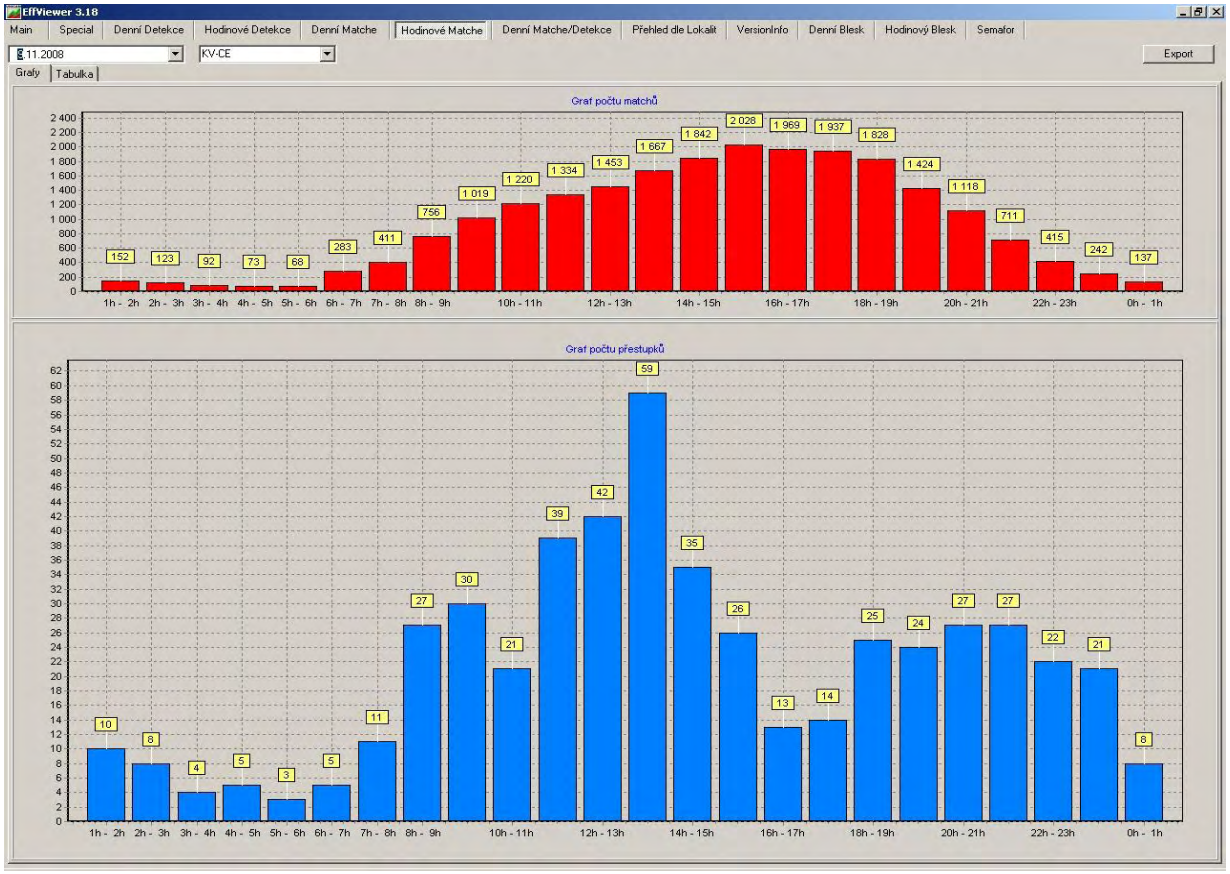
Stavba prototypů je standardní technikou pro validaci produktu v prostředí, ve kterém bude používán, a to před jeho finální výrobou. Prototyp může mít některé aspekty odlišné od finálního produktu (např. cena, rozměr), ale musí podporovat tu vlastnost, která je na něm ověřována. Pokud se pomocí jedné technologie ověřuje činnost jiné, hovoříme o emulaci. Např. FPGA obvody se používají pro emulaci např. mikroarchitektur procesorů, kdy lze výrazně urychlit validaci funkce i za cenu, že ne všechny parametry jsou takové, jaké má výsledný systém mít. (rychlost, příkon apod.). Platformy s FPGA byly v široké míře též použity pro stavbu prototypů vizuálních systémů popsaných v příloze.

Při testování je model, prototyp či produkt zkoušen, zda funguje správně. Testování se využívá jak při validaci (správné chování modelu), tak při verifikaci (správnost transformace) modelu. Jak výpočetní platformy uvedené v kapitole 4 , tak komplexní systémy (viz kapitoly 12 a 13) byly navrhovány s ohledem na snadnou testovatelnost. Veškeré subsystémy mají zabudovány tzv. self-testy, které zajišťují otestování základních vlastností (např. test paměti) před spuštěním aplikace. Pro potřeby testování propojení jednotlivých integrovaných obvodů na desce s plošnými spoji (angl. Boundary Scan), vnitřní struktury FPGA a emulace DSP integrováno rozhraní JTAG.

Pro návrh FPGA se též používají prostředky pro validaci v reálném čase pomocí ladění přímo na čipu, které probíhá na téměř plné pracovní frekvenci obvodu (např. nástroj ChipScope firmy Xilinx). Jedná se o doplnění sond a přídavné logiky (komparátory, registry, apod.), které umožní realizace funkcí, jako jsou logický analyzátor, analyzátor sběrnic či virtuální rozhraní. Díky tomu lze sledovat jak signály, tak komponenty uvnitř FPGA. Logické hodnoty jsou monitorovány na rychlosti rovné či blízké pracovní frekvenci obvodu a jsou přenášeny po analýzu do vhodného vizualizačního nástroje (např. Logic Analyzer firmy Xilinx). Výhodou je i možnost propojení se standardními měřicími přístroji.

I v případě, že byly využity veškeré techniky validace uvedené v předchozích odstavcích, je nezbytné mít ve funkčním systému zabudovány kontrolní mechanizmy. Základním způsobem, jak monitorovat správnou činnost systému za provozu, je vyžití hlídacích obvodů (angl. watchdog), tj. obvodu, který, pokud není periodicky nulován z aplikačního programu (např. v důsledku přetečení zásobníku), generuje reset systému.

V reálném systému, který musí pracovat nepřetržitě (24 hodin 365 dní v týdnu), je třeba neustále sledovat veškeré parametry, zda nevybočují z předefinovaných intervalů. Intervaly jsou dány buď na základě expertních znalostí, či stanoveny na základě statistik získaných za provozu systému. Na Obr. 35 jsou histogramy výskytu některých parametrů systému pro monitorování dopravy (viz kapitola 12), které spolu při správné funkci systému musí korelovat. Stupeň korelace je průběžně sledován. Pokud vybočí z daného rámce, může to znamenat poruchy systému a je to hlášeno servisnímu středisku.



Obr. 35 Monitorování chování systému za provozu

Mezi velmi účinné metody patří vkládání monitorovacích sond do běžících aplikací, které umožní jednak kontrolovat tvrzení (asserts) jak na straně SW, tak HW a vytváření statistik (histogramy, logy) o činnosti programu. V případě nesplnění nějakého tvrzení je pak možno on-line reagovat např. generováním výjimky pomocí přerušení a obsloužit ji deterministickým způsobem voleným s ohledem na jeho závažnost (logování, reset atd.). Průběžná kontrola statistik je zdrojem informací o chování systému za běhu a pomáhá odhalit i takové chyby, které jsou jiným způsobem prakticky neodhalitelné (nepředpokládaný vliv okolního prostředí ap.). U systému Unicam jsou uvedené techniky zabudovány na všech úrovních od kamer až po servery. Výsledky kontrol jsou průběžně hlášeny nadřazenému monitorovacímu systému, který je tak schopen je neprodleně hlásit operátorovi. Dále pak lze provádět analýzy logovaných záznamů pomocí skriptovacích jazyků, generovat statistiky apod. Pro ilustraci si uveďme chování systému, který je realizováno na základě informací generovaných v inteligentních kamerách systému Unicam (viz odstavec 4.8). V FPGA platformy DX6, která je v kameře vestavěna, jsou integrovány kontrolní obvody, které monitorují činnost kamery. Tyto údaje jsou s každým pořízem snímkem odesílány do nadřazeného systému, ve kterém se průběžně vyhodnocují a archivují. On-line kontroly zabezpečují např. detekci výpadku globální synchronizace systému (GPS). U systému pro měření

rychlosti jízdy vozidel se jedná o stav, při kterém se musí systém zrušit veškerá prováděná měření a přejít do nouzového režimu (je to dáno příslušnou normou). Příklad záznamu o činnosti kamery:

```
------------------------------------------------------------------------
------ D A T U M :   20080921                                 ------------
------------------------------------------------------------------------
-----------------------------------
Lokalita: 20080921_PCSD
Verze FW:
DSP:5.26 VIRTEX:1.11 FPS:21.8 <plat>
-----------------------------------
Lokalita: 20080921_PCSD2
Verze FW:
Nenalezen log pro 1. kameru lokality 20080921_PCSD2
{E} 21.09.2008 01:59:45 TIMECHECK: !!! diff: 999998 last good seq: 34686   imageID: 146178
Pocet vyskytu: 1
-----------------------------------
Lokalita: 20080921_praha_pd_o
Verze FW:
Crate:6.47 Cam:UDP FPGA:1.144 MCU:1.65
{E} 21.09.2008 01:59:45 TIMECHECK: !!! diff: 1000001 last good seq: 29678   imageID: 141409
Pocet vyskytu: 1
```

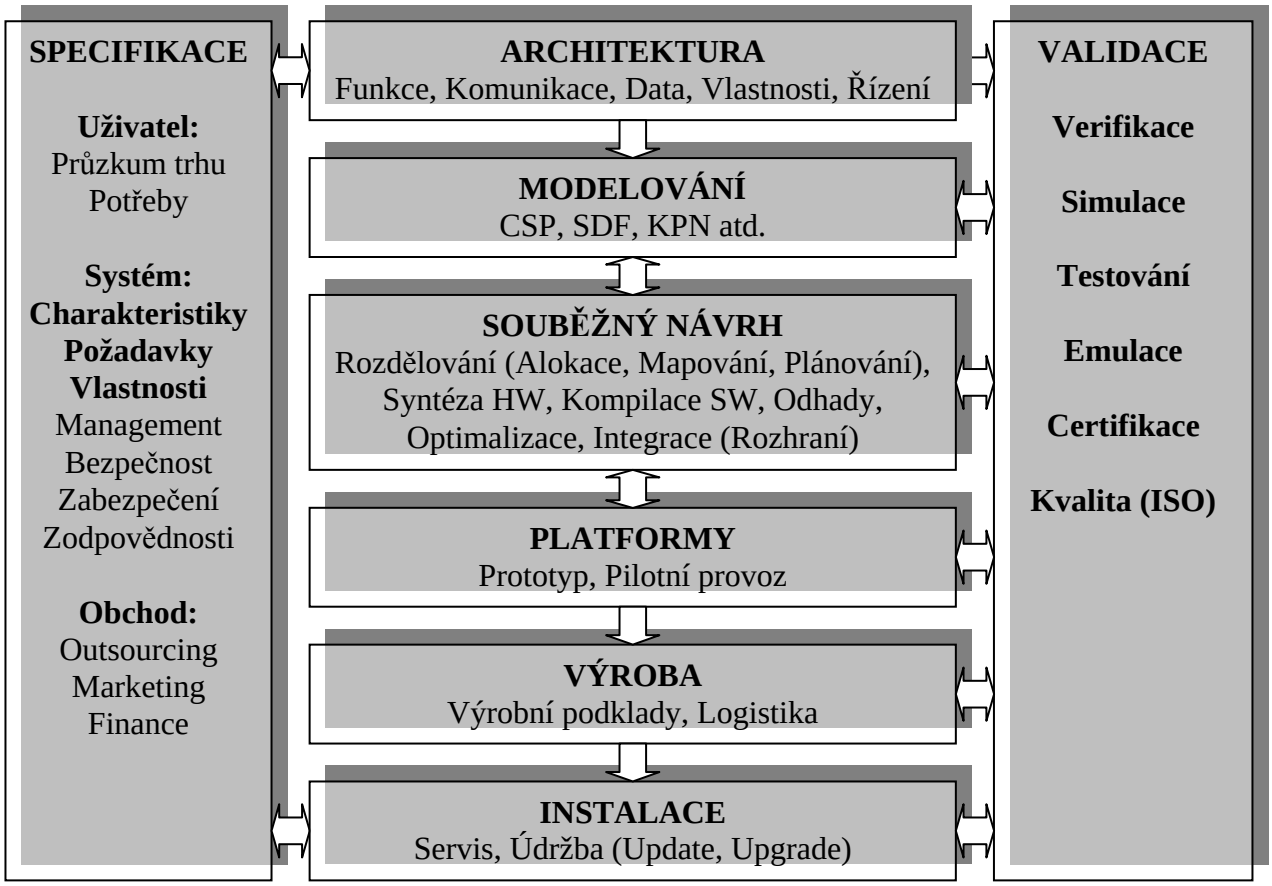
Informace ze záznamů se kontroluje jak on-line, tak při analýzy případné poruchy viz např. extrahovaná informace pomocí skriptu v jazyce PERL:

```
   ...
{M} 21.09.2008 01:58:06 Regular Aperture Info:  Aperture: 0  Exposure: 2000  Mean Image Value: 3
{M} 21.09.2008 01:59:06 Regular Aperture Info:  Aperture: 0  Exposure: 2000  Mean Image Value: 3
{D} 21.09.2008 01:59:12 asking:  current: 140717  asked: 140723  ((TS: 1446703))
{E} 21.09.2008 01:59:45 TIMECHECK: !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
{E} 21.09.2008 01:59:45 TIMECHECK: !!! diff:      1000001   last good seq: 29678   imageID:
141409
{D} 21.09.2008 01:59:45 TIMECHECK: !!! d-info:    1047883   486950320-485902437 (47882)
{D} 21.09.2008 01:59:45 TIMECHECK startup:   0   2008-09-20 23:59:46.296240
{M} 21.09.2008 01:59:45 [01479890] GPStime:  0005403b 00148277   000000fd   1
`$GPRMC,235946.059,A,5006.3790,N,01432.2702,E,0.00,48.21,200908,,*31            '
{D} 21.09.2008 01:59:45 TIMECHECK startup:   1   2008-09-20 23:59:46.344123
...
```

Řada testů lze provádět až na finálně instalovaném systému. Proto se v praxi standardně realizuje tzv. pilotní provoz prototypu v reálných podmínkách a po instalaci je každý systém testován ve zkušebním provozu.

9 SPECIFIKACE A ARCHITEKTURA

Návrh komplexních systémů je třeba provést s ohledem na jejich celý životní cyklus (viz Obr. 36), který začíná specifikací produktu a definicí jeho architektury. Při návrhu systémů, tak jak byly popsány v předchozích kapitolách, se předpokládalo, že jak jejich specifikace, tak architektura byly dány předem a návrh začínal až fází modelování. Pro HSC komplexních systémů je však třeba zohlednit i tyto úvodní fáze, neboť jednak se může specifikace měnit i během návrhu systému a dále je třeba mít jasně definovanou architekturu, která nám může usnadnit tvorbu nových aplikací. V souladu s výše uvedeným je i dobrá zkušenost s adaptivní metodikou návrhu, která klade důrazem na validaci všech fází a kroků návrhu s potřebami a požadavky zákazníka a přizpůsobovat jim, díky použití platforem s FPGA, nejen SW, ale i HW systému. Jednotlivé kroky budou přiblíženy na příkladu realizace systému pro monitorování dopravy Unicam – viz kapitola 12 .



Obr. 36 Životní cyklus systému

9.1 SPECIFIKACE

Před zahájením detailního návrhu systému je třeba mít jasnou specifikaci, která se provádí na základě důkladné analýzy potřeb uživatelů, průzkumu trhu, stanovení charakteristik systému a požadavků na systém. Je třeba definovat rozhraní a interakce mezi systémem a uživatelem. V případě, že systém bude mít více uživatelů, je třeba určit i vzájemné vztahy mezi nimi (např. systémy pro zjišťování dopravních přestupků jsou typicky v majetku města, ale provozuje je policie a je udržován pověřenou servisní organizací). Vlastnosti systému určují jednotlivé funkce užitečného a funkčního celku spolu s hlavními řídicími a datovými toky.

Koncepce systému definuje, jaké služby a funkce uživatelé potřebují, čeho se má systémem dosáhnout (vize), jací lidé budou systémem ovlivněni a budou jej ovlivňovat, jaký vývojový tým zajistí realizaci systému atd.

Charakteristiky systému

Na základě identifikovaných potřeb uživatelů a ujasněné koncepce systému se definují charakteristiky systému (vlastností, které bude výsledný systém mít), mezi které patří:

- Popis – jak bude systém vypadat, jak bude pracovat atd.
- Funkce – vyjmenování principiálních funkcí systému.
- Institucionální – identifikace institucí či úřadů, které budou nějakým způsobem dotčeny systémem či jej budou ovlivňovat.
- Organizační – kdo bude systém ovládat, kdo bude zpracovávat data atd.
- Sociální – budou koncoví uživatelé či veřejnost systém akceptovat?
- Právní – jaké zákony platí pro provoz systému, či jaké zákony může systém ovlivnit.
- Data – jaké informace bude systém potřebovat pro svůj provoz a jaká data bude generovat.
- Komunikace – hlavní komunikační kanály v rámci systému a s okolím.
- Bezpečnost – může systém za provozu někoho zranit?
- Zabezpečení – mohou ze systému uniknout citlivé informace? Může někdo systém neoprávněně užívat?
- Elektrické – zajištění napájecího napětí, příkonu, jištění atd.
- Elektromagnetická kompatibilita a odolnost proti rušení (EMC, EMI).
- Mechanické – rozměry, hmotnost, odolnost, atd.
- Nestandardní režimy činnosti – jak se systém chová v době upgrade atd.
- Údržba – je systém snadno přístupný? Jaká je četnost profylaktických prohlídek? Jsou použity nějaké komponenty s omezenou životností (např. baterie)? Způsob update a upgrade.
- Budoucí rozvoj – plánují se nějaké nové funkce v budoucnu a je na to architektura systému připravena?
- Finanční náklady – cena a přínosy systému. Kdo bude financovat jeho nákup?
- Provozní náklady – jaká je cena údržby a kdo ji bude platit?
- Rizika – existují nějaká rizika (cena, technické možnosti, atd.)?

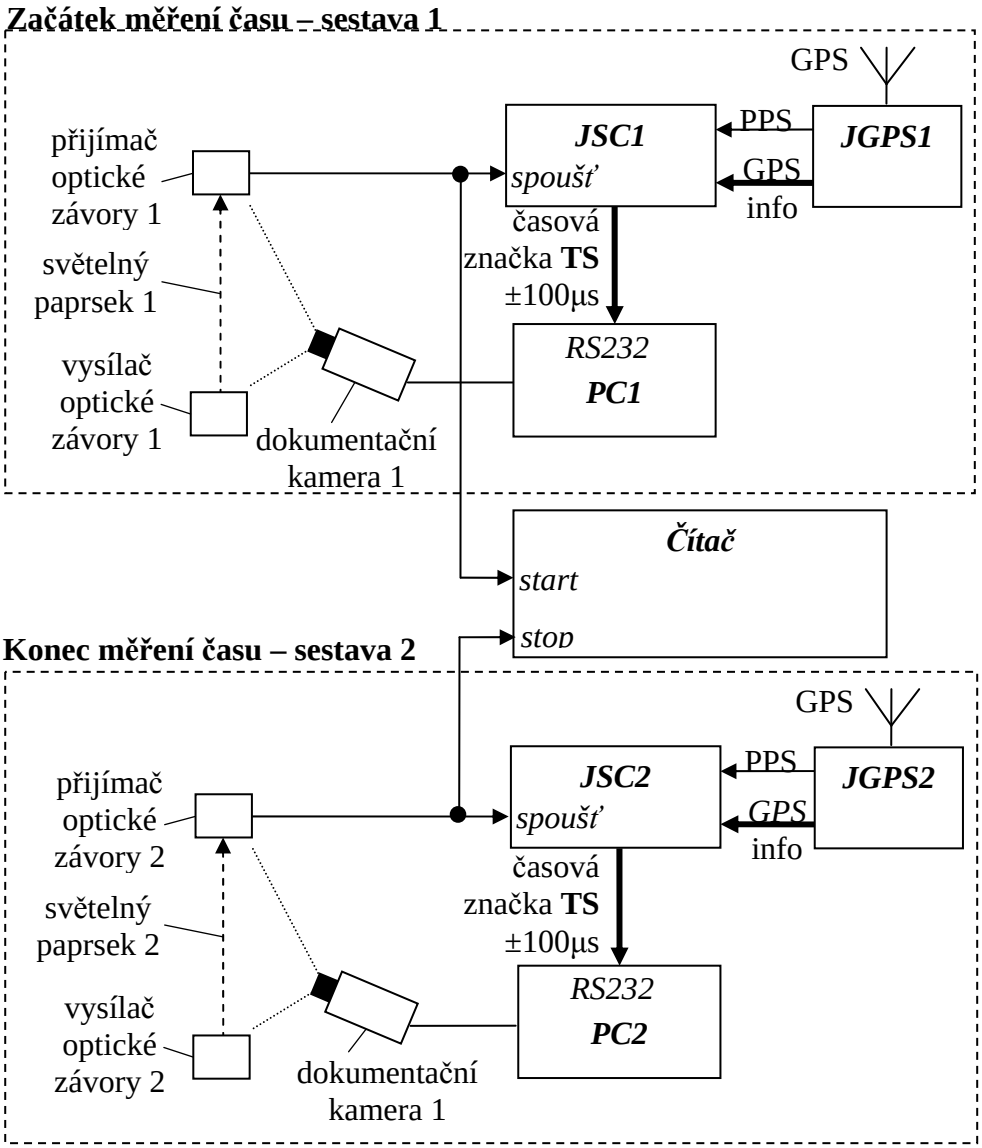
Požadavky na systém

Na základě charakteristik systému lze definovat požadavky, jež jsou formálním vyjádřením cílů, které má systém naplnit. Vychází z koncepce systému a charakteristik systému. Požadavky na systém je třeba validovat s potřebami uživatele a verifikovat s charakteristikami systému. I tato fáze je uživatelsky orientovaná. Je často velmi obtížné přesně identifikovat potřeby uživatele, který nemusí mít přesnou představu, co mu může např. nová technologie přinést. Často jsou potřeby uživatelů nereálné a protichůdné. Jasná definice a vyřešení všech nejasností je základem, neboť řada požadavků přímo ovlivňuje definici architektury systému. Charakteristiky systému říkají, co je požadováno, kdežto požadavky na systém říkají, jak toho bude dosaženo. Mezi typické požadavky na systém patří vlivy okolního prostředí, ve kterém bude systém pracovat a jak v něm bude systém pracovat. Je důležité, aby bylo jasně definované rozhraní architektury systému s okolím. Tedy to co je součástí architektury a co není a může tedy systém ovlivňovat nebo jím být ovlivněno. Zde patří např.:

- Otevřenost – použití standardních komponent.
- Inovace – systém umožňuje budoucí rozvoj (snadný update, upgrade).

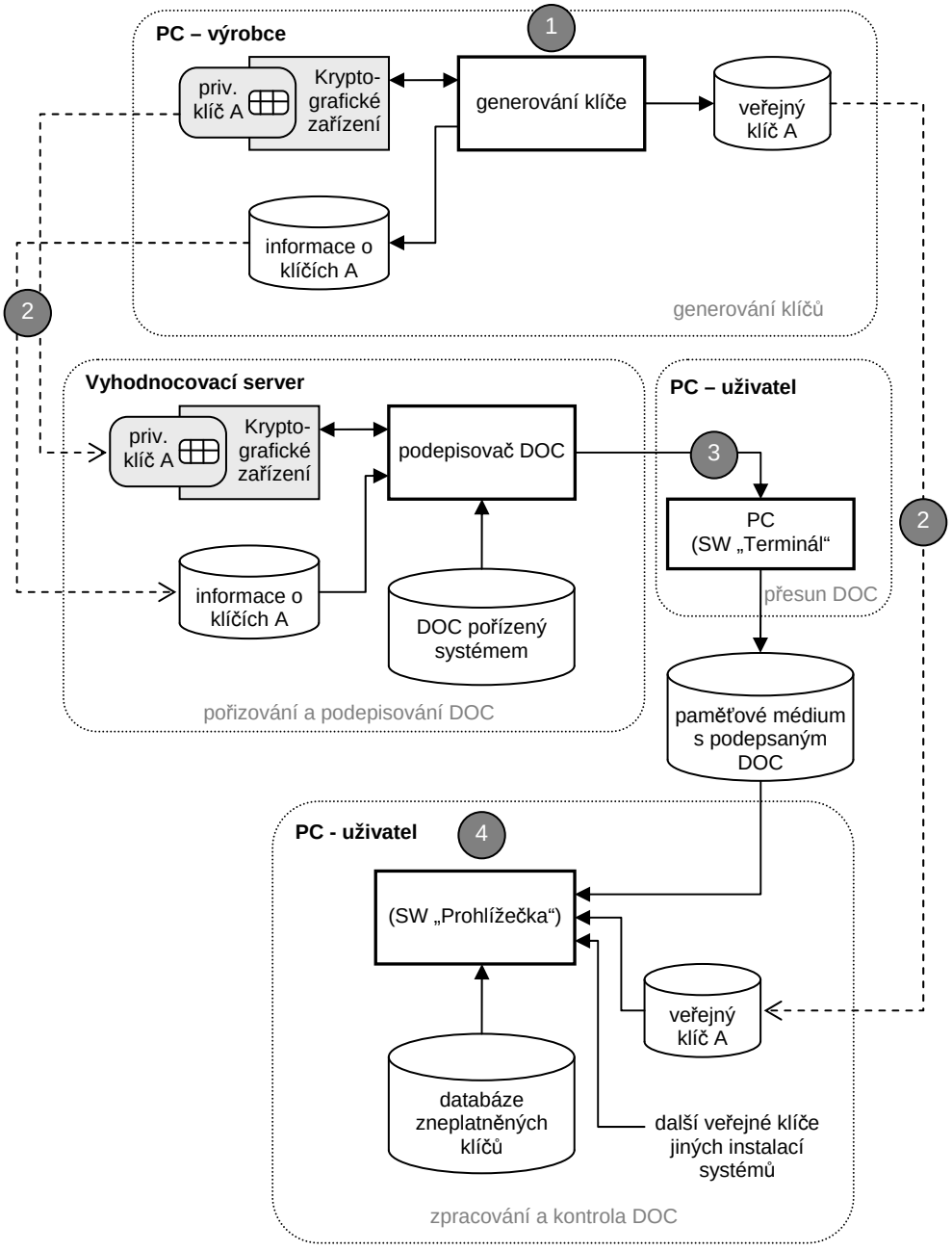
- Geografické – je umožněna rozšiřitelnost systému na větší území (speciální komunikační a napájecí infrastruktura atd.).
- Institucionální – omezující podmínky s hlediska zodpovědnosti za provoz atd.
- Finanční – pořizovací cena versus přínosy. Prodejní cena versus náklady na vývoj. Cena údržby.
- Sociální – akceptování systému uživateli a dotčenými stranami.
- Technické – dostupnost a možnosti technologií.
- Rizika – způsob omezení rizik.
- Infrastruktura – maximální kompatibilita se současnou infrastrukturou.

Některé požadavky na systém jsou odvozeny od jeho určení (zákonné normy atd.). Např. v době vývoje rychloměru pro měření úsekové rychlosti systému Unicam (viz kapitola 12) nebyly ještě vypracovány metodiky pro jeho certifikaci v souladu, která je u takových zařízení vyžadována zákonem. Jednalo se totiž o nový, do té doby v ČR nepoužitý princip. Na situaci bylo třeba reagovat tak, že se, díky flexibilitě architektury, návrh daného systému přizpůsobil potřebám ověřování jeho metrologických vlastností.



Obr. 37 Metodika certifikace systému

Dále byl vyvinut systém, který měří rychlost jízdy kontrolního vozidla (etalon) používaného Českým metrologickým institutem (ČMI) využíván pro ověřování i jiných rychloměrů instalovaných v terénu. Ve spolupráci s ČMI byla též navržena metodika, podle které je dnes v ČR prováděno ověřování všech rychloměrů pro měření rychlosti na dlouhých úsecích a byla zapracována do návrhu příslušné vyhlášky Ministerstva obchodu a průmyslu. Metodika certifikace metrologických vlastností silničního rychloměru je principielně ilustrována na Obr. 37. Při certifikaci jde o ověření přesnosti měření na hladině významnosti 99,8 %.



Obr. 38 Příklad dat v systému

Dalším příkladem vlivu zákonných norem na architekturu systému je práce s citlivými údaji. V systémech Unicam je ochraně dat věnována maximální pozornost. Architektura podporuje použití vhodných zabezpečovacích prostředků na všech úrovních práce s citlivými údaji (pořizování, zpracování, přenosu, archivace atd.). Každý subsystém a propojovací infrastruktura

jsou chráněny způsobem, který je volen s ohledem na charakter dat a jejich jednotlivé umístění. Jedná se např. o ochranu polohou, detektory a indikátory (plomba) neoprávněné vniknutí, zabezpečovací zařízení, monitoring datových toků, použití aplikačně specifických protokolů, šifrování, navazováním spojení je ze strany systémů (ne serveru), řízení hesel atd. O servisních pracích a přístupu k jednotlivým subsystémům jsou vedeny podrobné záznamy a přístup je regulován řízeným dokumentem (směrnicí). Příklad způsobu zabezpečení dat v systému je ilustrován na Obr. 38.

Vlastnosti systému definují hierarchii zodpovědností v systému (management, bezpečnost a zabezpečení). Při návrhu je vhodné oddělit tok dat (dataflow) v systému od toku jeho řízení (controflow). Jedná se o popisy definující vnitřní strukturu systému, což usnadňuje tvorbu rodin systémů s množstvím společných vlastností (snížení nákladů díky využití stejných komponent).

Vlastnosti systémů musí zajistit:

- Jednoduchost – pro jednoznačnou validaci uživateli a použití návrháři.
- Stabilitu – systém musí být stabilní (často po řadu let) a zároveň modifikovatelný pro adaptaci na prostředí a potřeby uživatele.
- Flexibilitu – spíše předepisovat než popisovat (snadná adaptace nových technologií atd.).
- Bezpečnost – identifikace a alokace zodpovědností.
- Zabezpečení – zajištění zabezpečení dat a ochrana soukromí.

9.2 ARCHITEKTURA

Při tvorbě komplexního systému je nezbytné předem definovat jeho architekturu. Architekturou rozumíme behaviorální a strukturní popis, který definuje kostru, okolo které jsou vyvíjeny systémy určité třídy. Po uvedení produktu do praxe pak architektura systému zajišťuje stabilní bázi pro jeho provoz. Architektura systému by měla být vytvořena ještě před zahájením konkrétního návrhu systému, a to na základě specifikace vytvořené na základě identifikace potřeb uživatelů. Jakmile je architektura definována, je většinou velmi drahé ji později měnit. Architektura systému tedy není ještě návrhem systému, ale popisem, který určuje základ pro jistou třídu systémů a tedy i množinu jejich návrhů. Popisuje všechny atributy třídy systémů a specifikuje struktury, které jsou fixní a ty, které mohou mít více instancí. Struktura systému je typicky definována jednotlivými částmi architektury (viz dále), které popisují konkrétní vlastnosti jako např.:

- Funkce – hierarchický popis struktury a chování systému.
- Řízení – popisuje tok řízení mezi subsystémy.
- Data – typy a oblemy dat, se kterými systém pracuje.
- Fyzická – dekompizice na subsystémy a komunikační kanály mezi nimi.
- Komunikace – popisuje tok dat mezi subsystémy, charakteristiky přenosových médií.
- Management – popisuje metody správy a údržby systému.
- Obchod – popisuje obchodní vztahy mezi dodavatelem systému, uživateli, správcem atd.

Tradiční návrhové techniky jsou založeny na předpokladu, že je možné získat vytvořit úplnou specifikaci a následně vytvořit produkt, který ji bude splňovat. V případě rozsáhlých systémů je však třeba mít možnost adaptovat jejich vlastnosti dle požadavků uživatelů. Pokud by architektura systémů byla neměnná, adaptace systému na lokální podmínky (např. lokalizace pro zahraniční trh s odlišnou legislativou) by mohla být velmi omezená či dokonce nemožná. Proto je třeba definovat architekturu fixně jen do jisté míry. Architektura systému musí na všech úrovních umožňovat evoluci systémů i po jeho nasazení v praxi. Flexibilita nejen že umožňuje implementaci nových funkcí na základě požadavků uživatelů, ale usnadňuje též provoz, údržbu (update, upgrade) a servis.

Zkušenosti ukazují, že pokud stávající systém rozšíříme (přidáme nové instalace, funkce ap.), objeví se nové problémy. Složitost systému výrazně roste s každým novým subsystémem, který

interaguje se zbytkem systému. Např. pokud máme v praxi nasazen jeden systém, pak jeho údržbu zvládne poměrně málo lidí. Pokud ale máme takových systémů instalovány desítky, pak je třeba řešit problémy spojené s centrálním vyhodnocením údajů, koordinací činností vícečlenného týmu, kapacitu datových úložišť, automatický monitoring funkce a vzdálená správa systému atd. Tyto konsekvence je třeba identifikovat a plánovat již v počátečních fázích návrhu architektury systému a ponechat prostor pro dostatečnou flexibilitu, která umožní adaptaci nových technologií a postupů, které ze života systému objeví.

Příkladem je systém synchronizace času v systémech Unicam: U rozsáhlých a geograficky velmi vzdálených systémů je jedním z nejdůležitějších parametrů jejich vzájemná synchronizace (jednotný čas). Technické řešení není snadné – nelze např. propojit všechny systémy synchronizačním kabelem (na delší vzdálenosti je to technicky nerealizovatelné díky zpoždění signálů a ceně). Použití např. bezdrátových sítí a síťových protokolů též nelze akceptovat (výpadky spojení, nedeterministické časové odezvy). Architektura systému umožnila implementaci velmi stabilní časové základy synchronizované na systém GPS s vícenásobnou kontrolou případných výpadků (s využitím FPGA a DSP). Tato funkce umožňuje realizaci např. systémů měření dojezdových dob atd. (viz kapitola 12), které by jinak prakticky nebylo možno realizovat, čímž se otevřel prostor pro tvorbu řadu unikátních aplikací.

Za života systému se může projevit řada nežádoucích a neočekávaných jevů, které ovlivní či znemožní jeho správnou funkci. Typicky se jedná o opotřebení, vliv okolního prostředí (střídání teplot, elektromagnetické záření) a vliv rozvoje systému (kapacita přenosových tras je fixní, ale díky rozšíření počtu instalací systému je třeba přenášet řádově větší objemy dat) či spolupráce s okolím (neočekávaný zásah operátora, selhání spolupracujícího systému atd.). Tyto jevy nelze vždy dopředu předvídat a je často velmi obtížné je identifikovat a odstranit. I zde však platí, že pokud je architektura systému navržena správně, může do jisté míry vliv opotřebení či prostředí omezit (orientace na striktní dodržování norem, používání standardizovaných a ověřených postupů, certifikace, robustní konstrukce atd.).

Udržovatelnost je důležitou vlastností komplexních systémů. Jedná se o proces, který probíhá po celou dobu životního cyklu systému. Udržovatelnost je dána úsilím, financemi, časem, vzděláváním a školením uživatelů, které je třeba, aby instalovaný systém přešel z počátečního funkčního stavu (instalace) do požadovaného užitečného stavu (rutinní provoz). Jelikož se požadavku uživatele mohou měnit na základě zkušeností s jeho provozem či změnou legislativy, mění se i požadovaný stav a je třeba k tomu při údržbě přihlédnout. Údržba tedy není jen servis v případě poruchy, ale jedná se o komplexní službu, která je součástí provozu systému jako takového. Flexibilní systém ještě nemusí být dobře udržovatelný, avšak flexibilita může umožnit lepší adaptaci a udržovatelnost systému dle požadavků uživatele (nové funkce) či vlivu prostředí (změna legislativy může implikovat organizační změnu při provozu systému). Součástí údržby jsou modifikace existujícího systému po jeho dodání uživateli. Jedná se o update a upgrade – korekce chyb, vylepšení vlastností (výkonnosti atd.) a adaptace na změny prostředí, ve kterém je používán (nově vzniklé požadavky uživatelů atd.).

U systémů popsaných v příloze bylo již při koncepci architektury počítáno s možností snadné údržby na několika úrovních. Jednak jsou jednotlivé komponenty plně programovatelné. Např. inteligentním kamerám systému lze vzdáleně přehrát veškerý firmware (pro FPGA, MPU i DSP). Zde se prokázalo, že důraz na použití výpočetních platforem s FPGA byla strategický významná volba, neboť je možno modifikovat jak SW, tak HW i u systému, který je instalován v terénu. Dále jsou veškeré komponenty systému propojeny s centrálním serverem, což umožňuje centralizovaný update a upgrade. Veškeré subsystémy mají též integrovány funkce pro on-line validaci a monitorování. Každá porucha a případné neočekávané chování systému je ihned detekováno, zaznamenáno a avizováno správci systému, který může neprodleně zahájit příslušný servisní úkon.

Při vývoji systému spolupracoval tým desítek lidí na všech úrovních od uživatelů, přes legislativní zabezpečení, proces certifikace, výzkum metod počítačového vidění, HSC, zajištění

výroby, servisu a údržby. Systémový přístup, jasně specifikovaná architektura systému a použití adaptivních metod návrhu s použitím technik HSC výrazně přispěly ke koordinaci a optimalizaci jednotlivých činností. Jedním z trendů současnosti je úzká specializace. S ohledem na budování komplexních heterogenních systémů může, však úzká specializace nestačí. Důležitá je zde role architekta systému, který musí být nejen chopen pohlížet na systém jako na celek při jeho návrhu, podobně a být schopen analyzovat vliv nežádoucích jevů na správnou funkci systému ve všech jeho aspektech.

9.3 SHRNUTÍ

Cílem specifikace a definice architektury systému je zajistit stabilní rámec pro tvorbu užitečných a funkčních systémů. Základem je funkční systém, tedy takový, ve kterém nejen že správně pracují všechny jeho subsystémy, ale že tyto subsystémy vzájemně kooperují tak, že celý systém pracuje, jak bylo požadováno uživatelem. Užitečný je takový systém, který přináší uživateli přínosy, které očekával a to po celou dobu životního cyklu. Užitečný systém nejen že funguje jak má, ale je též snadno použitelný, udržovatelný (snadná a levná údržba, vysoká spolehlivost atd.) a provozovatelný (nízký příkon, levný provoz komunikační infrastruktury apod.). Uvedené vlastnosti systému od sebe nelze odlišit, protože jen pokud jsou všechny v souladu s potřebami uživatele, můžeme hovořit o úspěšném produktu. Formálněji lze požadavky vyjádřit pomocí výkonnostních parametrů (klimatická odolnost, výpočetní výkon, kapacita datových úložišť, přenosové kapacity komunikačních kanálů, atd.) a kvalitativních parametrů (flexibilita, rozšiřitelnost, udržovatelnost, spolupráce s ostatními systémy, testovatelnost, bezpečnost, zabezpečení ap.).

Funkčnost a užitečnost se často, z úzkého zaměření na proces návrhu systému, nerozlišují, ale s ohledem na jeho následné použití v praxi, je nezbytné obojí zohlednit v celém životním cyklu produktu. Např. první počítače byly funkční (šlo s nimi počítat), ale rozměrné, málo výkonné, drahé a nespolehlivé. Teprve s technologickým pokrokem výroby integrovaných obvodů se jejich vlastnosti vylepšily natolik, že se staly užitečnými i pro běžného uživatele. Čím je funkční produkt užitečnější, tím má větší šanci uspět na trhu.

Složitost procesu návrhu komplexních výpočetních systémů je dána především obrovskou velikostí prostoru možných řešení, která roste s každým přidaným subsystémem a funkcí. S rostoucí složitostí systémů klesá míra jejich srozumitelnosti jak pro návrháře, tak pro uživatele. Jedním z důvodů nefunkčního systému může být právě nemožnost či neschopnost používat příliš složitý systém. Obrovský prostor možných řešení je dán tím, že použitelné výpočetní komponenty mají různou cenu, výkonnost, flexibilitu, spolehlivost, příkon, rozměry, složitostí atd. Dále platí, že s ohledem na propojení komponent existuje řada možností tvorby rozhraní mezi těmito komponentami. Při návrhu je třeba dodržet řadu omezujících podmínek na výsledný produkt jako je cena, výkonnost, příkon a především dobou návrhu. Složitost návrhu též roste se složitostí vývojových nástrojů a návrhových technik.

Mezi delší požadavky determinující návrh patří přenositelnost, bezpečnost, zabezpečení, uživatelské rozhraní, kompatibilita se standardy atd. Optimalizace systému pro jednu množinu požadavků často vede na neakceptovatelné vlastnosti v jiné. Např. architektura, která je navržena s ohledem pro vysoký výpočetní výkon ve všeobecných aplikacích, nemusí zároveň splňovat požadavky na životnost baterií a cenu.

Základním principem jak složitost omezovat je princip „keep-it-simple“, kdy se neimplementuje zbytečně to, co není bezprostředně požadováno pro uspokojení potřeb uživatelů. V řadě případů platí, že pokud nefunguje byť jen 1% systému, znamená to, že systém z hlediska uživatele nefunguje vůbec (např. systémy v letadle). Čím je tedy systém jednodušší, tím větší šanci má být plně funkční a užitečný. Zároveň je dobré systém částečně předimenzovat (výpočetní

výkon, kapacita paměťových, šířka pásma přenosových tras) jako rezervu pro budoucí update či dokonce upgrade systému nasazeného v praxi.

V praxi též platí pravidlo “1:10:100”, které popisuje úsporu nákladů, jichž lze dosáhnout, pokud se podaří případné chyby nalézt v úvodních fázích návrhu systému. Náklady na opravu chyby v počátku návrhu může stát 1/10 nákladů toho, co by stála oprava finálního produktu a 1/100 toho, co stojí oprava systému, který je již nasazen v praxi. Dobrá koncepce systému a vhodná metodika práce vede nejen snížení pravděpodobnosti systémových chyb, jejichž odstranění je nejdražší a často neproveditelné, ale též k jejich bezprostřednímu odhalování již při návrhu.

Rychlost, s jakou je možno nový systém uvést na trh, je často klíčovým parametrem tvorby ceny produktu. Zisk z prodeje produktu je totiž určen nejen rozdílem mezi prodejní a výrobní cenou (cenou návrhu se rozpočítává do jednotlivých výrobků), ale především podílem na trhu. Podíl na trhu určuje, vedle užitných vlastností produktu, okamžik jeho uvedení na trh a náklady, které musí uživatel platit za jeho provoz a údržbu. V případě brzkého (či jako vůbec prvního) uvedení produktu na trh je vyšší pravděpodobnost zisku díky většímu podílu na trhu a většímu prodeji. Cena návrhu představuje počáteční investici, která se musí vrátit co nejdříve. Čím déle musí investor čekat, tím větší musí být návratnost investic[13].

[13] Jiná situace je např. u státních zakázek, kdy jsou nejdůležitějším parametrem náklady na celou dobu životnosti systému (tedy včetně údržby) a dále pak u kritických aplikací (ochrana zdraví, základní výzkum, vojenství), kde jsou určujícími parametry např. bezpečnost, spolehlivost apod.

10 ZÁVĚR

Cílem práce byl výzkum a praktické použití pokročilých technik návrhu aplikačně specifických výpočetních systémů, které umožňují optimalizaci často protichůdných kritérií kladených na jejich implementaci, v rámci daném dostupnými technologiemi a společenskou poptávku. V tomto kontextu je práce zaměřena na výzkum, vývoj a použití výpočetních platforem s programovatelným hardwarem při tvorbě inženýrských děl s důrazem na techniky souběžného návrhu hardware a software jak vestavěných systémů, tak komplexních vizuálních systémů. V práci jsou též uvedeny postřehy a zkušenosti z inženýrského přístupu k dané problematice zasazeného do formálního rámce. Pro ověření pokročilých technik návrhu v oblasti tvorby vestavěných systémů se autor podrobněji zabýval zkoumáním potenciálu technologií programovatelného hardware a jeho využitím jak ve výzkumu a ve výuce, tak při vývoji a návrhu vysoce výkonných výpočetních platforem. Mezi systémy s největším společenským dopadem, které jsou prezentovány v této práci, patří akcelerátor pro číslicové zpracování signálů DX6, inteligentní kamera UnicamD, síťová platforma COMBO6 a platformy pro číslicové zpracování signálů (DX64, UNI1P a DSPX). V oblasti komplexních systémů se je uvedeno využití zkoumaných návrhových postupů při implementací platformy pro tvorbu multifunkčních inteligentních kamerových dopravních systémů Unicam, v jejímž rámci byly instalovány desítky systémů se stovkami inteligentních kamer v ČR i zahraničí a obratem téměř 200 mil. Kč. Technické řešení je patentováno, chráněno užitnými vzory a certifikováno v ČR i zahraničí. Systém má velký společenský dopad (pokles dopravních přestupků a nehod s fatálními důsledky v desítkách procent, velké ekologické dopady). Dalším významným inženýrským dílem, na jehož tvorbě se autor aktivně podílel a který využívá výsledky daného výzkumu, je platforma pro tvorbu vizuálních systémů kontroly výrobků na výrobních linkách CVS, v rámci které byly instalovány stovky systémů s tisíci kamerami v ČR i zahraničí s obratem přes 150 mil. Kč.

Rychlost, s jakou je uveden produkt na trh, určuje podstatnou položkou ceny produktu a právě souběžný návrh HW a SW (HSC) může výrazně přispět k jejímu zkrácení. V práci jsou prezentovány zkušenosti a aplikací těchto technik jak na úrovni návrhu výpočetních platforem, tak s jejich zobecněním pro tvorbu komplexních výpočetních systémů. Důraz je kladen na maximální opětovné použití v praxi osvědčených komponent a flexibilitu nejen SW, ale též HW díky použití programovatelného hardware. Dalším přístupem usnadňujícím návrh komplexních systémů, které autor dlouhodobě aplikuje, je znovupoužití již jednou navržených a vyrobených výpočetních systémů – platforem. V případě platforem je důležitá flexibilita použitého hardware tak, aby bylo platformu možno přizpůsobit široké škále aplikací. Hlavní důraz při návrhu je kladen spíše na spojení jednotlivých HW a SW částí a jejich interakce, než na syntézu komponent samotných. Základní charakteristikou většiny tradičních HSC přístupů je to, že návrh je orientován na tvorbu nových, vysoce optimalizovaných systémů (např. ASIC). V praxi se tento přístup aplikuje jen zřídka, neboť např. návrh nového ASIC čipu si můžeme dovolit jen v případě masové produkce. Při tvorbě praktických aplikací se však ukazuje, že se většinou jedná o postupnou evoluci funkcí systémů od jednodušších ke složitějším. Využívají se stávající návrhy, které se adaptují podle potřeby a rozšiřují o nové funkce. V tomto kontextu se jako výhodné ukázalo aplikování adaptivních a inkrementálních návrhových technik, kdy se modifikuje či doplňuje jen část systému a bezprostředně následuje jejich validace.

Plně automatizovaný přístup k HSC není prozatím efektivní a je pravděpodobně nedosažitelný. Na základě dlouholetých zkušeností s koncepcí, návrhem a vedením do praxe řady komplexních systémů se nejlépe osvědčila kombinace „inženýrského“ přístupu a ověřených vědeckých postupů a metod. Jde o využití toho nejlepšího, co každá technika nabízí a nalezení vhodného způsobu jak je vzájemně integrovat. Tato metoda je pak aplikována v celém procesu návrhu na všech úrovních.

Výsledkem práce je shrnutí dlouhodobých zkušeností s tvorbou komplexních výpočetních systémů, mezi které patří: Chování systémů je třeba modelovat na co nevyšší úrovni abstrakce za

použití takových výpočetních a komunikačních modelů, které jsou přirozené pro vyjádření daného chování a přitom usnadňují jeho implementaci (viz GALS modely). Je vhodné využít heterogenní přístup, ve kterém se při konstrukci komplexních systémů systematicky kombinují různé modely. Pro prohledávání prostoru možných řešení, za použití optimalizačních technik, je nezbytná detailní inženýrská znalost dané problematiky, která může vést na nalezení vhodné kriteriální funkce nejlépe vyjadřující vlastnosti daného problému a heuristik, které mohou prohledávání urychlit. Složitost integrovaných obvodů roste podstatně rychleji než produktivita práce při návrhu výpočetních systémů. Vzhledem k tomu, že prakticky použitelná a komerčně úspěšná řešení jsou spíše doménově a platformě orientovaná se však ukazuje, že motorem vysoké produktivity práce je znovupoužití již jednou vytvořených a řádně otestovaných komponent. Návrhové metody se tedy dnes musí orientovat též na integraci hotových, než jen na automatizovanou tvorbu nových komponent. Jednotlivé komponenty lze pak navrhovat tradičními metodami, u kterých návrhář pečlivě optimalizuje jednotlivé parametry důležité pro danou aplikační doménu. V některých případech se sice plně automatickou tvorbou komponent může dosahovat lepších výsledků, než při jejich tvorbě člověkem, ale jejich návrh (prozatím) vyžaduje expertní znalosti z oblasti umělé inteligence a je vhodný jen pro obvody malé složitosti. Problematiku návrhu v praxi použitelných obvodů též nelze studovat pouze na syntetických a testovacích problémech. Pro implementaci systému je výhodné používat heterogenní platformy, které se skládající z různých výpočetních prostředků a kde každá komponenta optimálně vykonává určitou část výpočtu. Díky složitosti systémů se některé problémy dají vyřešit až v průběhu návrhu a je tedy nezbytné využívat adaptivní metody návrhu. Důležitou charakteristikou praxí ověřeného přístupu k tvorbě komplexních systémů je orientace na tvůrčí tým. Inženýr má často znalosti, které je obtížné, či nemožné, formálně zapsat. Řada rozhodnutí při procesu návrhu je intuitivního charakteru a volba řešení dána s jistou pravděpodobností. Systémový návrh vyžaduje nejen spojení SW a HW inženýrských metod, ale i orientaci na vědecké metody a nástroje aplikovatelné v daném ekonomickém a časovém rámci a vyžaduje multidisciplinární přístup s orientací na uspokojování reálných potřeb uživatelů při dodržení daných podmínek. Podrobněji jsou některé z výše uvedených závěrů uvedeny na konci jednotlivých kapitol.

Cíle práce byly naplněny úspěšným ověřením účinnosti zkoumaných inženýrských a vědeckých metod a postupů při tvorbě reálných systémů s velkým společenským dopadem. Předmětem dalšího výzkumu bude hlubší studium možností modelování na systémové úrovni, s ohledem na optimalizaci výsledné implementace dle různých kritérií, včetně jejich ověření při tvorbě a použití rekonfigurovatelných výpočetních platforem v praxi.

10.1 PODĚKOVÁNÍ

Díky složitosti jsou prezentovaná inženýrské díla kolektivními pracemi, na kterých se různým způsobem podílely desítky spolupracovníků. Mezi nejbližší kolegy, jež přispěli k tvorbě popisovaných systémů a publikovaných prací, a kterým bych rád touto cestou poděkoval za spolupráci, patří (v abecedním pořadí): Bardas Radomír, Beran Vítězslav, Beszedes Marian, Bia Tomáš, Bittner Martin, Bryan Luděk, Čejka Rudolf, Dulík Tomáš, Filip Vojtěch, Hegar Antonín, Herout Adam, Honec Jozef, Honec Petr, Kalová Ilona, Kořenek Jan, Kovář Milan, Kučera Leoš, Lexa Matej, Lizstwan Marek, Martínek Tomáš, Mastný Miroslav, Novotný Jiří, Richter Miloslav, Přibyl Tomáš, Soušek Antonín, Šustek Jiří, Tupec Petr, Valenta Pavel, Zanca Dalibor, Zemčík Pavel, Zezulka Jan a další.

Hlavní poděkování patří mé ženě Hance, dětem Honzovi, Leničce a Martinovi za trpělivost a podporu.

Tato práce byla vytvořena za finančního přispění rozvojového programu "Podpora akademických pracovníků – habilitace".

11 LITERATURA

[1] AMDAHL, Gene. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. AFIPS Conference Proceedings, Vol. 30. Atlantic City, N.J., USA, AFIPS Press : 1967, pp. 483-48. Reprint, dostupné na WWW: <http://www.ieee.org/portal/site/sscs/menuitem.f07ee9e3b2a01d06bb9305765bac26c8/index.jsp?&pName=sscs_level1_article&TheCat=2171&path=sscs/07Summer&file=Amdahl67.xml>.

[2] ANTOS, D.; REHAK, V.; KORENEK, J. Hardware Router's Lookup Machine and its Formal Verification; In Proceedings of 3rd International Conference on Networking (ICN'04). Gosier, Guadeloupe, France : 2004.

[3] ANTOŠ, D.; KOŘENEK, J. Hardware Router's Lookup Machine and its Formal Verification, In Proceedings of the ICN'2004 Conference. Gosier, Guadeloupe, French : 2004.

[4] BRYAN, L.; FUČÍK, O. FPGA Implementation of a Reconfigurable License Plate Detection Method. In Proceedings of the Engineering of Reconfigurable Systems and Algorithms, ERSA 2007. Las Vegas, USA: 2007, pp. 1-4. ISBN 1-60132-026-4.

[5] BUCK, J., T. Scheduling dynamic dataflow draphs with bounded memory using the token flow model. Technical Report UCB/ERL 93/69. Department of EECS, University of California, Berkeley, CA 94720, 1993.

[6] CAMEA, spol. s r. o. Způsob měření doby průjezdu a zařízení k provádění tohoto způsobu. Původce: HONEC, J.; FUČÍK, O.; RICHTER, M.; VALENTA, P.; ZEMČÍK, P. Patentový spis 295326. 2005.

[7] CLARKE, E., M.; EMERSON, E., A.; SISTALA, A., P. Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst., vol. 8, no. 2. New York, NY, USA : ACM, 1986, pp. 244—263. ISSN0164-0925.

[8] CRHA, L.; FUČÍK, O.; DRÁBEK, V. HW-Based Object Detection Method for Traffic Monitoring, In Proceedings of the 6th Electronic Circuits and Systems Conference, ECS 2007. Bratislava, SK: 2007, pp. 93-96. ISBN 978-80-227-2697-9.

[9] CRHA, L.; FUČÍK, O.; DRÁBEK, V. Image Filter Implementation in FPGA used for License Plate Recognition. In Proceedings of 38th International Conference MOSIS'04. Ostrava, Czech Republic: 2004, pp. 169-174. ISBN 80-85988-98-4.

[10] CRHA, L.; FUČÍK, O.; ŠUSTEK, J. Environment for Hw/Sw Codesign of Embedded Systems, In Proc. of 8th IEEE Design and Diagnostic of Electronic Circuits and Systems Workshop, Sopron, HU: 2005, s. 236-240. ISBN 9639364487.

[11] CRHA, L.; FUČÍK, O.; ZEMČÍK, P.; DRÁBEK, V.; TUPEC, P. Inter Chip Communicating System with Dynamically Reconfigurable Hardware Support. In Proceedings of the 6th IEEE International Workshop on DDECS 2003. Poznaň, Poland: 2003, pp. 311-312. ISBN 83-7143-557-6.

[12] DeBARDELABEN, A., J.; MADISETTI, V., K.; GADIENT, A. Incorporating Cost Modelign in Embedded-System Design. IEEE Design & Test of Comp. 1997, pp. 24-35.

[13] DeMICHELI, G.; ERNST, R.; WOLF, W. Readings in Hardware/Software Co-Design. Morgan Kaufmann Publishers, 2002, pp. 293-426. ISBN 1-55860-702-1.

[14] ELSSAMADISY, A. Agile Adoption Patterns: A Roadmap to Organizational Success. Addison-Wesley Professional, 2008. ISBN 10: 0321514521.

[15] FUČÍK, O. ANN Implementation Based on Bit-Serial Stochastic Computing that Exploits FPGA's RAM Capability. In Proceedings of the 6th School of Neural Networks Theory and Applications. Sedmihorky: 1994, pp.173-180.

[16] FUČÍK, O. FLEDGE: A Flexible Digital Computer Architecture. In Proceedings of the TUB Workshop '96. Brno, Czech Republic: 1996, pp. 267-268.

[17] FUČÍK, O. Hardware/Software Applications Evaluation Using Flexible Microcomputer System MMX. In Proceedings of the International Conference Real-Time ’95. Ostrava, Czech Republic: 1995, pp. 150-153.
[18] FUČÍK, O. HW/SW Co-Design of Routers, In Sborník konference EuroOpen. Strážnice, Czech Republic: 2003.
[19] FUČÍK, O. Neural Network Implementation Using FPGAs. In Proceedings of the International Conference Real-Time ’95. Ostrava, Czech Republic: 1995, pp. 93-96.
[20] FUČÍK, O. Rekonfigurovatelné vestavěné systémy : doktorská disertace. Brno : VUT v Brně, ÚIVT, FEI, 1997, 69 s.
[21] FUČÍK, O.; ZEMČÍK, P.; LOKER, D.; FORD, R.; WEISSBACH, R. A Flexible DSP Hardware Platform for LVDT Signal Conditioning. In Proceedings of the 4th International Symposium on Advanced Electromechanical Motion Systems – ELECTROMOTION '01. Bologna, Italy: 2001.
[22] FUČÍK, O.; ZEMČÍK, P.; TUPEC, P.; CRHA, L.; HEROUT, A. The Networked Photo-Enforcement and Traffic Monitoring System. In: Proceedings of Engineering of Computer-Based Systems, ECBS 2004. Los Alamitos, USA: 2004, pp. 423-428. ISBN0-7695-2125-8.
[23] FUČÍK, Otto. Flexibilní mikropočítačový systém MMX. Sdělovací technika 4, 1995. 1995, str. 152-153.
[24] GUSTAFSON, J., L. Reevaluating Amdahl's law. Commun. ACM, vol. 31, no. 5. New York, NY, USA : ACM, 1988, pp. 532-533. ISSN0001-0782.
[25] HEMANI, A.; MEINCKE, T.; KUMAR, S.; POSTULA, A.; OLSEN, T.; NILSSON, P.; OBERG, J.; ELLERVEE, P.; LINDQVIST, D. Lowering power consumption in clock by using globally asynchronous locally synchronous design style. In Proceedings of the 36th ACM/IEEE Conference on Design Automation. New Orleans, Louisiana, USA : 1999, pp. 873-878.
[26] HERRING, C. Microprocessors, Microcontrollers, and Systems in the New Millenium. IEEE Micro, vol. 20, no. 6. 2000, pp. 45-51.
[27] HOARE, C., A., R. Communicating Sequential Processes. Communications of the ACM 21. 1985, pp. 666—677. Dostupné na WWW: http://www.usingcsp.com/cspbook.pdf.
[28] CHIODO, M.; GIUSTO, P.; HSIEH, H.; JURECSKA, A.; LAVAGNO, L.; SANGIVANNI-VINCENTELLI, A. Synthesis of mixed software-hardware implementations from cfsm specifications. In International Workshop on Hardware-Software Co-design. 1993.
[29] CHANDRAKASAN, A., P.; SHENG, S.; BRODERSEN, R., W. Low-power CMOS digital design. Solid-State Circuits, IEEE Journal of Volume 27, Issue 4. 1992, pp. 473 – 484.
[30] CHOW, C. T.; TSUI, L., S., M.; LEONG, P., H., W.; LUK, W. WILTON, S. Dynamic Voltage Scaling for Commercial FPGAs. In Proceedings of the International Conference on Field-Programmable Technology. 2005. pp. 173-180.
[31] IBM And Xilinx Prepare For Production Of First 90nm Chips On 300mm Wafers [online]. Dostupné na WWW : http://www-03.ibm.com/press/us/en/pressrelease/392.wss.
[32] IBM Moves Moore's Law into the Third-Dimension [online]. Dostupné na WWW: http://www-03.ibm.com/press/us/en/pressrelease/21350.wss.
[33] ITRS 2001 edition. The International Technology Roadmap for Semiconductors: 2001. Dostupné na WWW: http://public.itrs.net.
[34] KILBURN, T. Životopis. The University of Manchester, USA: 2001. Dostupné na WWW: <http://www.computer50.org/mark1/kilburn.html>.
[35] KORF, R., E. Recent Progress in the Design and Analysis of Admissible Heuristic Functions. Book Series Lecture Notes in Computer Science, Springer, Volume 1864/2000 : 2000, pp. 45-55. ISSN 1611-3349.

[36] LEE, E., A.; MESSERSCHMITT, D., G. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Trans. Comput. 36, 1. IEEE Computer Society : 1987, pp. 24-35. ISSN0018-9340.
[37] LEE, Edward, A. Overview of the Ptolemy Project [online]. Technical Memorandum No. UCB/ERL M03/25. Berkeley, CA, USA: 2003. Dostupné na WWW: <http://ptolemy.eecs.berkeley.edu/publications/papers/03/overview/overview03.pdf>.
[38] LEXA, M.; MARTÍNEK, T.; BECK, P.; FUČÍK, O.; VALLE, G.; ZARA, I. Genomic PCR simulation with hardware-accelerated approximate sequence matching, In Proceedings of the 21st European Conference on Modelling and Simulation, ECMS 2007. Praha, Czech Republic: 2007, pp. 333-338.
[39] LIU, M.; CAI, M.; TAUR, Y. Scaling Limit of CMOS Supply Voltage from Noise Margin Considerations. In International Conference on Simulation of Semiconductor Processes and Devices. 2006, pp. 287-289.
[40] MAHAJAN, R.; BROWN, K.; ATLURI, V. The Evolution of Microprocessor Packaging [online]. Intel Technology Journal, 3rd Quarter 2000. Intel Corporation: 2000. Dostupné na WWW: <http://developer.intel.com/technology/itj/q32000/articles/art_1.htm>.
[41] MARTIN, G.; LAVAGNO, L.; LOIUS-GUERIN, J. Embedded UML: a merger of real-time UML and codesign. In CODES 2001. Copenhagen : 2001, pp.23-28.
[42] MARTÍNEK, T. ; LEXA, M.; BECK, P.; FUČÍK, O. Automatic Generation of Circuits for Approximate String Matching, In Design and Diagnostics of Electronic Circuits and Systems, DDECS 2007. Krakow, Poland: 2007, pp. 203-208. ISBN 1-4244-1161-0.
[43] MARTÍNEK, T.; KOŘENEK, J.; FUČÍK, O.; LEXA, M. A Flexible Technique for the Automatic Design of Approximate String Matching Architectures. In Proceedings of the Design and Diagnostics of Electronic Circuits and systems DDECS 2006. Washington, DC, USA : IEEE Computer Society, 2006, pp. 83-84. ISBN 1-4244-0185-2.
[44] McGRATH, D.: Gartner Dataquest analyst gives ASIC, FPGA markets clean bill of health [online]. [Cit. 2008-08-17]. <http://www.eetimes.com/conf/dac/showArticle.jhtml?articleID=164302400>.].
[45] Moore's Law 40th Anniversary Press Kit [online]. [Cit. 2008-08-17]. Dostupné na WWW: http://www.intel.com/pressroom/kits/events/moores_law_40th/index.htm?iid=tech_moores law+body_presskit.
[46] NOVOTNÝ, J.; ANTOŠ, D.; FUČÍK, O. Project of IPv6 Router with FPGA Hardware Accelerator. In Proceedings of the Field-Programmable Logic and Applications, FPL 2003. Edit. by Cheung P.Y.K.; Constantinides G.A.; de Souza J.T. Berlin-Heidelberg: Springer, Lecture Notes in Computer Science 2778, 2003, pp. 964-967. ISBN 3-540-40822-3.
[47] NOVOTNÝ, J.; FUČÍK, O.; ANTOŠ, D. Liberouter – New Way in IPv6 Routers. In Proceedings of the 2nd International Conference ICETA 2003. Kosice, Slovak Republic: Elfa, 2003, pp. 153-158. ISBN 80-89066-67-4.
[48] NOVOTNÝ, J.; FUČÍK, O.; BARDAS, R. Schematic of COMBO-4MTX Card. In Techniká zpráva 13/2003, CESNET [online]. 2003 [cit. 1.11.2008]. Dostupné z WWW: <http://www.cesnet.cz/doc/techzpravy/2003/combo4mtxschematic/combo4mtxschematic.pdf>.
[49] NOVOTNÝ, J.; FUČÍK, O.; BARDAS, R. Schematic of COMBO-4SFP Card. In Techniká zpráva 12/2003, CESNET [online]. 2003 [cit. 1.11.2008]. Dostupné z WWW: <http://www.cesnet.cz/doc/techzpravy/2003/combo4sfpschematic/combo4sfpschematic.pdf>.
[50] NOVOTNÝ, J.; FUČÍK, O.; BARDAS, R. Schematics and PCB of COMBO6. In Techniká zpráva 14/2002, CESNET [online]. 2002 [cit. 1.11.2008]. Dostupné z WWW: <http://www.cesnet.cz/doc/techzpravy/2002/combo6/combo6.pdf>.

[51] Occam Reference Manual. SGS-THOMSON Microelectronics Limited : 1995. Dostupné na WWW: < http://www.wotug.org/occam/documentation/oc21refman.pdf >.
[52] OLUKOTUN, K.; HAMMOND, L. The Future of Microprocessors. Queue 3, 7. New York, NY, USA : ACM, 2005, pp.26-29. ISSN 1542-7730.
[53] PARKS, T. M. Bounded Schedule of Process Networks : PhD thesis. University of California at Berkeley: 1995. Dostupné na WWW: <http://www.eecs.berkeley.edu/Pubs/TechRpts/1995/ERL-95-105.pdf>.
[54] PETERSON, D., A.; HENNESSY, J., L. Computer Organization and Design: The Hardware/Software Interface. 2005, Morgan Kaufmann Publishers. ISBN 1-55860-604-1.
[55] POLLACK, F. J. New microarchitecture challenges in the coming generations of CMOS process technologies (keynote, abstract). In Proceedings of the 32nd Annual ACM/IEEE international Symposium on Microarchitecture. Haifa, Israel, November 16–18. 1999. Washington, DC, USA : IEEE Computer Society, 1999 [cit. 2008-11-01]. Dostupné na WWW: http://research.ac.upc.edu/HPCseminar/SEM9900/Pollack1.pdf.
[56] RAJE, S. ASICs won't die, but they will be comatose [online]. [Cit. 2008-08-17]. Dostupné na WWW: <http://www.eetimes.com/showArticle.jhtml?articleID=17408263>.
[57] ROLRF, Ernst. Codesign of Embedded Systems: Status and Trends. IEEE Design and Test of Computers, vol. 15, no. 2. Los Alamitos, CA, USA : IEEE Computer Society Press, 1998, pp. 45-54.
[58] SANGIOVANNI-VICENTELLI, A.; MARTIN, G. Platform-Based Design and Software Design Methodology for Embedded Systems. In Proc. IEEE Design & Test of Computers, vol. 18. no. 6. 2001, pp. 23-33.
[59] SEKANINA, L. Evolvable Components: From Theory to Hardware Implementations. SpringerVerlag: 2004. ISBN3540403779.
[60] SMITH, C., U.; FRANK, G., A.; CUARDRADO, J., L. An Architecture Design and Assessment Systems for Software/Hardware Codesign. In Proceedings of the 22nd Design Automation Conference. Los Alamitos, USA : IEEE Computer Society Press,1985, pp. 417-424.
[61] SOUŠEK, Antonín. Rychlý vývoj se systémem MMX a jazykem Modula-2. Sdělovací technika 5,1995. 1995, str. 210-211.
[62] Stanford Research Inst. Electronically Controlled Microelectronic Cellular Logic Array. WAHLSTORM, S, E. Patent US3473160. 1969.
[63] STEFANOV, T.; ZISSULESCU, C.; TURJAN, A.; KIENHUIS, B.; DEPRETTE, E. System Design Using Kahn Process Networks: The Compaan/Laura Approach. In Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1. Washington, DC, USA : IEEE Computer Society, 2004. ISBN0-7695-2085-5-1.
[64] TEEHAN, P.; GREENSTREET, M.; LEMIEUX, G. A Survey and Taxonomy of GALS Design Styles. IEEE Design and Test of Computers, vol. 24, no. 5. 2007, pp. 418-428.
[65] TURLEY, J. The Two Percent Solution. Embedded. Com [online]. 2002-12-18 [cit. 2008-08-17]. Dostupné na WWW: <http://www.embedded.com/story/OEG20021217S0039>.
[66] VAHID, F.; GONG, J.; GAJSKI, D., D. A Binary-Constraint Search Algorithm for Minimazing Hardware during Hardware /Software Partitioning. In Proceedings of the Conference on European Design Automation. Grenoble, France : 1994, pp. 214-219. ISBN0-89791-685-9.
[67] Virtex-5 FPGA Configuration User Guide [online]. Dostupné na WWW: http://www.xilinx.com/support/documentation/user_guides/ug191.pdf.
[68] Virtex-5 FPGA Data Sheet: DC and Switching Characteristics [online]. 2008. Dostupné na WWW: <http://www.xilinx.com/support/documentation/data_sheets/ds202.pdf>.

[69] WEXELBLAT, R., L. The consequences of one's first programming langure. In Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems. New York, NY, USA : ACM, 1980, pp. 52-55. ISBN 0-89791-024-9.
[70] WHITTEN, J; BENTLEY, L. Systems Analysis and Design Methods. McGraw-Hill/Irwin, 2005. ISBN 10: 0073052337.
[71] World Semiconductor Trade Statistics (WSTS). [cit. 2008-08-17]. Dostupné na WWW: <http://www.wsts.org>.
[72] YEN, Ti-Yen.; WOLF, Wayne. Performance Estimation for Real-Time Distributed Embedded Systems. IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 11, 1998. Los Alamitos, CA, USA: IEEE Computer Society, 1998. pp. 1125-1136. ISSN 1045-9219.
[73] ZEMČÍK, P.; FUČÍK, O.; HONEC, J.; VALENTA, P. Visual Analysis on the Production Line. In Proceedings of the 10th Scandinavian Conference on Image Analyses. Lappeenranta, Finland: 1997, pp. 308-310.
[74] ZEMČÍK, P.; HEROUT, A.; BERAN, V.; FUČÍK, O.; SCHIER, J. Reconfigurable Image Processing Architecture. International Journal on Graphics, Vision and Image Processing (GVIP). Special Issue on Applicable Image Processing. 2006, pp.7-12. ISSN1687-3998.
[75] ZEMČÍK, P.; HEROUT, A.; BERAN, V.; POTÚČEK, I.; FUČÍK, O.; HONEC, J.; RICHTER, M; KALOVÁ, I.; LISZTWAN, M. Image Processing in Traffic Applications. International Journal on Graphics, Vision and Image Processing (GVIP). Special Issue on Applicable Image Processing. 2006, pp.7-12. ISSN1687-3998.
[76] ZEMČÍK, P.; HEROUT, A.; CRHA, L.; FUČÍK, O.; TUPEC, P. Particle rendering engine in DSP and FPGA. In Proceedings of Engineering of Computer-Based Systems, ECBS 2004. 2004, pp. 361- 368. ISBN: 0-7695-2125-8.

Vybrané publikace autora, které se vztahují k tématu této práce: [4], [6], [8-11], [15-23], [38], [42], [43], [46-50] a [73-76].

12 PŘÍLOHA 1 – VIZUÁLNÍ SYSTÉMY V DOPRAVĚ

V této příloze je uveden příklad komplexního a komerčně úspěšného systému s velkým společenským dopadem. Jedná se o telematický systém pro monitorování dopravy Unicam (výrobce firma Camea, spol. s r.o.). V následujících odstavcích je uveden popis architektury platformy Unicam a příklady jednotlivých produktů, které na jejím základě vzniky. Popis byl vytvořen s použitím interních materiálů firmy Camea a [www.unicam.cz].

12.1 PALTFORMA UNICAM

Zvýšení bezpečnosti a plynulosti silničního provozu, snížení škodlivých emisí a zlepšení dopravní obslužnosti jsou vážné problémy, se kterými se již dnes potýká většina měst po celém světě a jejichž dopad bude v budoucnu výrazně narůstat. Jedním ze způsobů, jak tento problém řešit, je použití tzv. inteligentních telematických dopravních systémů (angl. Intelligent Transportation Systems – ITS). Tyto systémy zahrnují řadu technologií, jejichž cílem je učinit dopravu bezpečnější, efektivnější a ekologičtější. ITS jsou systémy, které využívají „inteligenci umístěné na vozovky a do vozidel“ [http://www.etsc.be]. Při jejich návrhu je třeba zabezpečit jak tvorbu všech subsystémů, tak adresovat všechny aspekty jejich použití po dobu celého životního cyklu. Díky komplexnosti ITS systémů je pro jejich vývoj a zavedení do praxe nezbytný multidisciplinární týmový přístup v těsné spoluprácí s uživatelem.

V současné době se stále více používá vizuálních systémů k monitorování dopravní situace. S rozvojem technologií elektronických kamer, zvyšováním výkonu výpočetní techniky a především s rozvojem metod automatizovaného zpracování obrazové informace, se stále více projevují snahy o úplnou či částečnou automatizaci procesu sledování dopravní situace, detekci a záznam nestandardních situací nebo přestupků. Kamer se např. využívá k měření rychlosti vozidel, ke zjištění obsazenosti, rychlosti a struktury proudů vozidel na rychlostních komunikacích. Využívá se jich též k detekci dopravních přestupků, jako jsou vjezdy do zákazů nebo průjezdy křižovatkou na červenou. Častým požadavkem je též registrace státní poznávací značky (zpoplatněné oblasti – mýto, parkoviště či přechody hranic a pátrací akce policie).

Platforma Unicam je určena pro monitorování dopravy s využitím programových a technických prostředků počítačového zpracování obrazu. Systém snímá obraz z kamer, v reálném čase jej vyhodnocuje dle řady kritérií, komprimuje, ukládá a přenáší. Pro dosažení vysoké přesnosti detekce vozidel, se obrazová informace zpracovává pomocí algoritmů umělé inteligence na specializovaných výpočetních prostředcích. Architektura systému Unicam tvoří rámec pro návrh řady multifunkčních ITS systémů, ve kterých lze informace získané při zpracování obrazu z kamer použít pro více účelů. Jednotlivé systémy mohou být též vzájemně propojeny a integrovány.

Integrace

Většina komerčně dostupných vizuálních systémů pro monitorování dopravy nabízí izolovaně určité specifické funkce (např. detekce kolon, průměrnou rychlost dopravních proudů, detekce nehod a některé též sledování pohybu jednotlivých vozidel) bez vyššího stupně integrace, což omezuje možnosti jejich užití v rámci integrovaného ITS.

Architektura systému Unicam je navržena s ohledem na propojení lokálních, izolovaných ITS systémů, do jednoho integrovaného celku, což umožňuje automatizované získávání kvalitativně nových informací o charakteru a parametrech dopravy a účinné vynucování dodržování dopravních předpisů. Jedná se o propojení jednotlivých kamerových systémů s centrálním serverem, na kterém je instalován databázový, analytický a vizualizační software. Vytvoření sítě kamerových systémů pracujících v reálném čase není triviální problém, neboť obrazová údaje představují takový objem dat, že jejich přenos klade často nesplnitelné nároky na komunikační infrastrukturu. V řadě případů nelze použít např. vysokokapacitních optických sítí k přenosu

obrazu, protože by jejich vybudování byl příliš investičně a organizačně náročné. Často používaná komprese obrazu však znamená jeho degradaci a často i ztrátu informací, které jsou pro jednoznačnou identifikaci vozidel nezbytné. Bezdrátové komunikační sítě též nejsou schopny přenášet obraz v reálném čase v požadované kvalitě. Proto je nezbytné automatické zpracování obrazové informace v místě jejího pořízení (u kamer) a následný přenos jen releventní informace k dalšímu zpracování. Kamera s výpočetní jednotkou a příslušnými detekčními algoritmy pak tvoří „senzor". K tomuto účelu je nezbytný vývoj a implementace detekčních a rozpoznávacích metod počítačového vidění a jejich zpracování v reálném čase (tímto požadavkem byla motivována potřeba návrhu výkonných výpočetních platforem, viz kapitola 4). Pro zjištění přítomnosti vozidla v zorném poli kamery pak využívá videodetekce, která pracuje tak, že příslušný algoritmus hledá v obraze vozidla na základě určitých charakteristických příznaků. Typicky se jedná o přítomnost poznávací značky a její následné automatizované čtení (např. pro pátrání po odcizených vozidlech). Dále je v praxi třeba zjišťovat skladbu dopravního proudu (rozpoznání třídy vozidla – osobní, nákladní apod.). Pro potřeby pátrání po odcizených vozidlech je výhodné zjišťovat typ vozidla (např. Škoda Fabia).

Centrální databáze naměřených údajů umožňuje jejich vizualizaci pomocí tzv. zátěžových map pro potřeby operativního řízení dopravy operátorem. Dále je třeba možno zprostředkovávat tyto údaje ve vhodné formě účastníkům silničního provozu např. na informačních tabulích. Analytický software vyhodnocuje dopravní situaci z globálního pohledu celé aglomerace a výsledky předává řídicímu systému dopravy a operátorům. Naměřená dopravní data jsou dále využita pro nové způsoby řízení dopravního toku prostřednictvím proměnného dopravního značení.

Dopady

Dle dostupných informací nám není známo žádné technické řešení inteligentních kamerových systémů pro ITS, které by dosahovalo takového stupně integrace jednotlivých technologií. Výsledný systém umožňuje kvalitativně úplně nový pohled na dopravní situaci a při jeho širokém nasazení může významně přispět bezpečnějším silnicím, nižším emisím a hluku a k lepšímu řízení a plánování dopravy. Vzhledem k neexistenci obdobného systému představuje platforma Unicam zcela „novou konstrukční koncepci" systému[14] pro zjišťování dopravních dat kamerovými systémy s cílem budování multifunkčních ITS.

Zkušenosti ukazují, že instalací ITS systémů lze snížit nežádoucí důsledky dopravy řádově o desítky procent. Platí, že informační a navigační systémy jsou jedním z pilířů ITS. Ovšem bez přesných, včasných a strukturovaných informací nelze ITS systémy budovat. V získávání těchto dat začínají dnes hrát klíčovou roli inteligentní kamerové systémy, jejichž příkladem jsou systémy Unicam. Zklidňování dopravy nelze uskutečnit bez účinného vynucování dodržování dopravních předpisů. Kamerové systémy zde opět hrají důležitou úlohu.

Podobně jako bezpečnost dopravy, tak i kvalita ovzduší a hluk na území měst jsou v současné době žhavým tématem. Evropské a národní právní předpisy, které jsou v platnosti, stanovují limity pro hluk a emise na území měst. Doprava představuje významný problém pro kvalitu ovzduší a nákladní doprava ve městech se na tomto velmi podílí. V současnosti se problémy staly již tak vážnými, že zpomalují ekonomický rozvoj městských aglomerací. Je známo, že např. snížení rychlosti vozidel na povolenou hranici nejen, že snižuje riziko dopravních nehod, ale může přispět i k tomu, aby hladiny hluku a emisí vyhověly prahovým hodnotám. Na základě zkušeností s používáním kamerových systémů pro měření rychlosti plyne, že ve všech lokalitách, ve kterých jsou systémy Unicam instalovány, došlo k výraznému poklesu rychle jedoucích vozidel (o 50 – 70%). Úměrně tomu se snížil počet fatálních dopravních nehod s (viz odezvy v médiích odstavec 12.7), klesly emise a hluk.

[14] Lze tedy konstatovat, že tento nový přístup řadí projekt do inovačního řádu 7 „druh".

Infrastruktura

S rozvojem rozsáhlých ITS systémů řešení je spojena řada problémů. Především se jedná o vytvoření sítě kamer, které sledují dopravní situaci v celé aglomeraci. Jejich vzájemným propojením je vytvořena síť, která umožňuje výrazné zvýšení užitné hodnoty jednotlivých systémů díky synergickému efektu. Data, získaná v jednotlivých lokalitách, lze totiž využít pro globální pohled na dopravní situaci v celé oblasti. Globální pohled otevírá prostor pro účinnou optimalizaci řízení dopravy. Jak již bylo řečeno, problémem je zajištění potřebné komunikační a napájecí infrastruktury. Architektura systému Unicam je navržena s ohledem na možnost využití komunikačních kanálů omezené šířky (komprese, řízení událostmi) a dále pak s ohledem na nízký příkon (optimalizace výpočetních jednotek s využitím FPGA, DPS atd.). Díky tomu se podařilo využít bezdrátových přenosů veškerých dat mezi jednotlivými kamerovými systémy a centrálním serverem. Napájení systémů se provádí ze sloupů veřejného osvětlení – v noci se dobíjí baterie, ze kterých systém ve dne pracuje. A konečně díky nízké hmotnosti a kompaktní konstrukci se jednotlivé systémy instalují na sloupy veřejného osvětlení a není tedy třeba žádných stavebních povolení apod. Uvedené charakteristiky systému, se ukázaly jako klíčové při rozvoji systému.

Činnosti

Pro ilustraci činností, které byly při tvorbě ITS systémů realizovány, si uveďme jen informativní přehled:

- Analýza možností využití možností metod počítačového zpracování obrazu.
- Komunikace s uživateli o možnostech instalace a integrace kamerových systémů.
- Analýza stavu stupně využití kamerových systémů u potenciálních zákazníků integrujících ITS technologie.
- Analýza technických parametrů a cen konkurenčních systémů, analýza možností vývoje nových technologií případně jejich licencování.
- Definice rozhraní mezi jednotlivými ITS technologiemi.
- Zabezpečení datové komunikace s centrálními dopravně-inženýrskými servery.
- Realizace rozsáhlé sítě ITS systémů a jejich připojení k centrálnímu serveru.
- Návrh a implementace robustních a efektivních komunikačních protokolů pro přenos údajů zjištěných ITS systémy s ohledem na všechny funkce a typy údajů.
- Provoz systému, zjištění vlastností a následnou optimalizaci technického řešení.
- Výzkum, vývoj a implementace nových metod počítačového zpracování signálů pro detekci a rozpoznávání objektů v dopravě (kategorie vozidla, barva vozidla, typ vozidla, skladba a parametry dopravního proudu, detekce neočekávaných událostí).
- Návrh a implementace vhodných datových struktur a databází pro efektivní ukládání a vyhledávání dat pořízených ITS systémy a jejich poskytování uživatelům.
- Vybudování databázového serveru.
- Vývoj a integrace analytického software (např. měření dojezdových dob, liniové a adaptabilní řízení dopravy, preference vozidel, dynamická navigace, dopravní průzkumy, informování řidičů a identifikaci vozidel na základě čtení poznávacích značek a rozpoznání kategorie či typu vozidla).
- Vývoj a implementace vizualizačního software pro ergonomické zprostředkování dopravní situace uživatelům (řízení a plánování dopravy, dopravní průzkumy, zpracování přestupků, krizové řízení, dynamická navigace, ap.).
- Vývoj a integrace vizualizačního software pro zpracování informací z přestupkových systémů, tvorba zátěžových map, záznam průjezdu identifikovaného vozidla.
- Vývoj a výroba inteligentních kamer ITS systémů.
- Údržba systémů (on-line monitoring, update, upgrade)

Aplikace

Architektura platformy Unicam umožňuje adaptaci pro celou řadu aplikací. Způsob využití je dán zvoleným HW (FPGA) a SW (DSP, PC) a může být např. následující: detekce průjezdu vozidel křižovatkou na červenou, měření průměrné rychlosti v úseku, měření dopravních dat, záznam situace na vozovce či video dohled. Mezi možné aplikace též patří automatizované identifikaci průjezdu vozidel danými místy, založené na principu automatického rozpoznáváni registračních značek, např. pro potřeby studií dopravního inženýrství, hledání odcizených vozidel, výběru mýtného či měření dojezdových dob.

V současnosti je Unicam na českém trhu nejrozsáhlejší multifunkční inteligentní ITS pro monitorování dopravy. Pro ilustraci uveďme krátký přehled původních produktů, které jsou na platformě Unicam postaveny:

- První kamerový systém pro detekci jízdy na červenou (UnicamREDLIGHT) v ČR, 1995.
- První kamerový systém pro měření úsekové rychlosti (UnicamVELOCITY) v ČR, 2003.
- Velmi kvalitní automatizované čtení poznávacích značek vozidel (UnicamLPR), 2004.
- První kamerový systém pro on-line pátrání po odcizených vozidlech (UnicamSCAN) v ČR, 2004.
- První kamerový systém pro měření dojezdových dob v ČR, 2006.
- Nová generace automatizovaných laserových rychloměrů s řadou unikátních funkcí (UnicamLIDAR), 2006.
- Sběr dopravních údajů, 2007.
- Telematické dohledové systémy s videodetekcí, 2008.
- Měření a vizualizace dojezdových dob v městské aglomeraci (UnicamTRAVELTIME), 2008.

Celkově bylo instalováno několik stovek inteligentních kamer v desítkách komplexních multifunkčních kamerových systémů po celé ČR a v zahraničí. V následujících odstavcích je uveden stručný popis některých významných aplikací s velkým společenským dopadem.

12.2 MĚŘENÍ ÚSEKOVÉ RYCHLOSTI

Jedním z původních výsledků výzkumu a vývoje v oblasti komplexních vizuálních systémů je silniční rychloměr pro měření úsekové (průměrné) rychlosti (MÚR) vozidel, která projedou předem vymezeným měřicím úsekem na vozovce. Jedná se o původní český výrobek (obchodní název UnicamVELOCITY) firmy CAMEA, spol. s r.o. a též první certifikovaný systém svého druhu[15]. Systém je rutinně používán policií a právními orgány již od roku 2003. Systém MÚR je používán pro zjišťování dopravních přestupků překročení rychlosti. Pro použití pro přestupkové řízení pořizuje rychloměr snímky vozidel, na kterých je vidět jak vozidlo a jeho SPZ/RZ, tak tvář řidiče a to i za zhoršených světelných podmínek. Bez uvedených náležitostí je obtížné či nemožné přestupky prokázat. Dále jsou přestupky elektronicky podepsány, čímž je zajištěno, že nemohou být nekontrolovaně modifikovány. Hlavní přidanou užitnou vlastností, oproti „klasickým“ rychloměrům, které měří rychlost jen v jednom místě – řezu vozovky (např. radary), je skutečnost, že se měří průměrná (matematicky správně tzv. střední) rychlost jízdy vozidla daným úsekem vozovky. U řezového měření okamžité rychlosti, kde řidiči typicky zpomalí v místě měření a za ním opět zrychlí. Zde musí dodržovat předepsanou rychlost v celém měřeném úseku. Tím se vynucuje dodržování povolené rychlosti v celém úseku což má vynikající preventivní účinky.

[15] Technické řešení je chráněno Úřadem průmyslového vlastnictví jako užitný vzor.

Popis

Od první instalace v roce 2003 prošel systém MÚR řadou inovačních změn. Současné řešení představuje již druhou generaci zohledňující jak technologický pokrok, tak především zkušenosti s jeho použitím policií a správními orgány. Na základě potřeb uživatelů byla v průběhu let zapracována řada nových funkcí a vlastností. Výsledkem vývoje je produkt, který byl vyvinut na „míru“ potřebám uživatele.

Registrační značka (RZ) a státní poznávací značka (SPZ) je považována za jediný průkazný identifikační prvek vozidla. V systému je maximalizována průkaznost přestupků díky viditelné SPZ/RZ i tváři řidiče, snímáním optimalizované sekvence přestupkových snímků a primárním tříděním snímků na základě čitelnosti SPZ/RZ, atd. Sporné a neprůkazné přestupky představují hlavní překážku při efektivním nasazení systému MÚR a jejich eliminace je tím, co určuje jeho kvalitu. Výsledkem dlouholetého vývoje je též pokrytí celé problematiky od zjištění přestupku, přes jeho zpracování, až po projednání s přestupcem – vše automatizovaně. Zdokumentované přestupky jsou jednoznačně identifikovány místem, datem, hodinou jejich spáchání. Okamžik spáchání přestupku je určen s vysokou přesností a stabilitou. Dále jsou přestupky elektronicky podepsány a při přenosu šifrovány, nelze je tedy modifikovat či zneužít.

Díky automatizovanému čtení SPZ/RZ se kamery využívají i pro pátrání po odcizených vozidlech. Dále lze do systému pro MÚR integrovat další videodetekční algoritmy sběr dopravních dat, detekci kolon či zastavených vozidel atd.

Princip činnosti

Činnost rychloměru je založena na definici rychlosti, jehož podstatou je měření doby průjezdu motorového vozidla měřicím úsekem vozovky, který má vyměřenou minimální délku. Rychloměr pak vypočte průměrnou rychlost vozidla *v*, jako podíl délky měřicího úseku Δs k změřené době průjezdu Δt podle vztahu $v = \Delta s / \Delta t$. Doba průjezdu měřeného vozidla Δt měřicím úsekem vozovky Δs se vypočítá jako rozdíl času vjezdu tohoto vozidla do měřicího úseku a času jeho výjezdu z tohoto úseku. Ze snímků, pořízených kamerami, které snímají začátek a konec měřícího úseku, se pomocí jednotky synchronizace času vytvoří ve vyhodnocovacím serveru tzv. referenční snímky. Využívá se při tom videodetekční počítačový program, který doby vjezdu a výjezdu automaticky určí a přiřadí na jednotlivé snímky.

Pro dosažení udané přesnosti rychloměru při maximální rychlosti měřených vozidel, musí mít měřicí úsek vozovky určitou minimální délku. Správnost měření doby průjezdu je zajištěna přesnou časovou synchronizací. Vypočtená průměrná rychlost vozidla je spolu s názvem místa měření, datem měření, časem výjezdu vozidla z měřícího úseku, identifikací jízdního pruhu, maximální povolenou rychlostí, délkou měřícího úseku a dobou průjezdu měřícím úsekem, zobrazena na referenčním snímku, pořízeném při výjezdu vozidla z měřicího úseku.

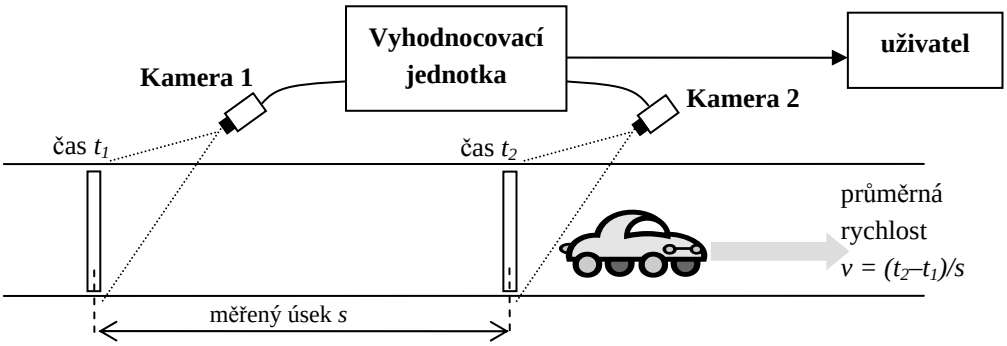
Systém rychloměru pracuje zcela automaticky. Následující parametry měření lze dálkově ovládat a nastavovat: zapnutí/vypnutí měření, nastavení aktuální maximální povolené rychlosti, hodnoty rychlosti klasifikované jako přestupek. Vlastní měření průměrné rychlosti probíhá zcela bezobslužně a nelze jej ovládacími prvky nikterak ovlivnit. Jeho správnost je zaručena tím, že vzdálenost měřicích míst (délka měřicího úseku) je změřena s vyžadovanou přesností a oba snímky jsou opatřeny časovými značkami z přesné časové základny.

Použitím elektronických kamer pro detekci vozidla na začátku a na konci měřícího úseku je také zaručeno, že rychloměr je pasivní, nevysílá žádné signály a je tedy prakticky nemožné jeho použití předem detekovat a jeho činnost ovlivňovat běžnými technickými prostředky. Konstrukce a prostorové umístění jednotlivých částí rychloměru je navrženo tak, aby byla vždy změřena minimální průměrná rychlost daného vozidla. Technickými prostředky a počítačovým zpracováním jsou vytvořeny takové podmínky, že nemůže dojít k poškození řidiče, tím, že by byla naměřena průměrná rychlost vyšší, než kterou ve skutečnosti jel. Konstrukce systému, vnitřní

logika měřícího procesu a ochranná opatření také zajišťují, že pokud je rychloměr použit v souladu s provozní dokumentací, nemůže být indikovaná rychlost připsána jinému vozidlu. Rychloměr též zruší výsledek měření, pokud nelze vozidlo jednoznačně identifikovat na základě jeho registrační značky, například při její nečitelnosti v důsledku znečištění apod.

Rychloměr je konstruován pro trvalé používání v kteroukoli roční dobu. Pro případ snížené viditelnosti může být vybaven na začátku i na konci měřicího úseku osvětlovací jednotkou.

Doba průjezdu vozidla Δt měřicím úsekem se určí z rozdílu časů $t_2 - t_1$ (časových značek) dvou referenčních snímků téhož vozidla pořízených na začátku s_1 (v čase t_1) a na konci s_2 měřícího úseku (v čase t_2). Princip činnosti je znázorněn na Obr. 39.



Obr. 39 Princip měření úsekové rychlosti

Zjištění přítomnosti vozidla v referenčním snímku funguje tak, že se ve snímcích hledá jednoznačný identifikační znak vozidla SPZ/RZ automatickou analýzou těchto snímků pomocí videodetekčního software, implementovaného pomocí algoritmů počítačového vidění a umělé inteligence.

obr. 1: Referenční snímky vozidla

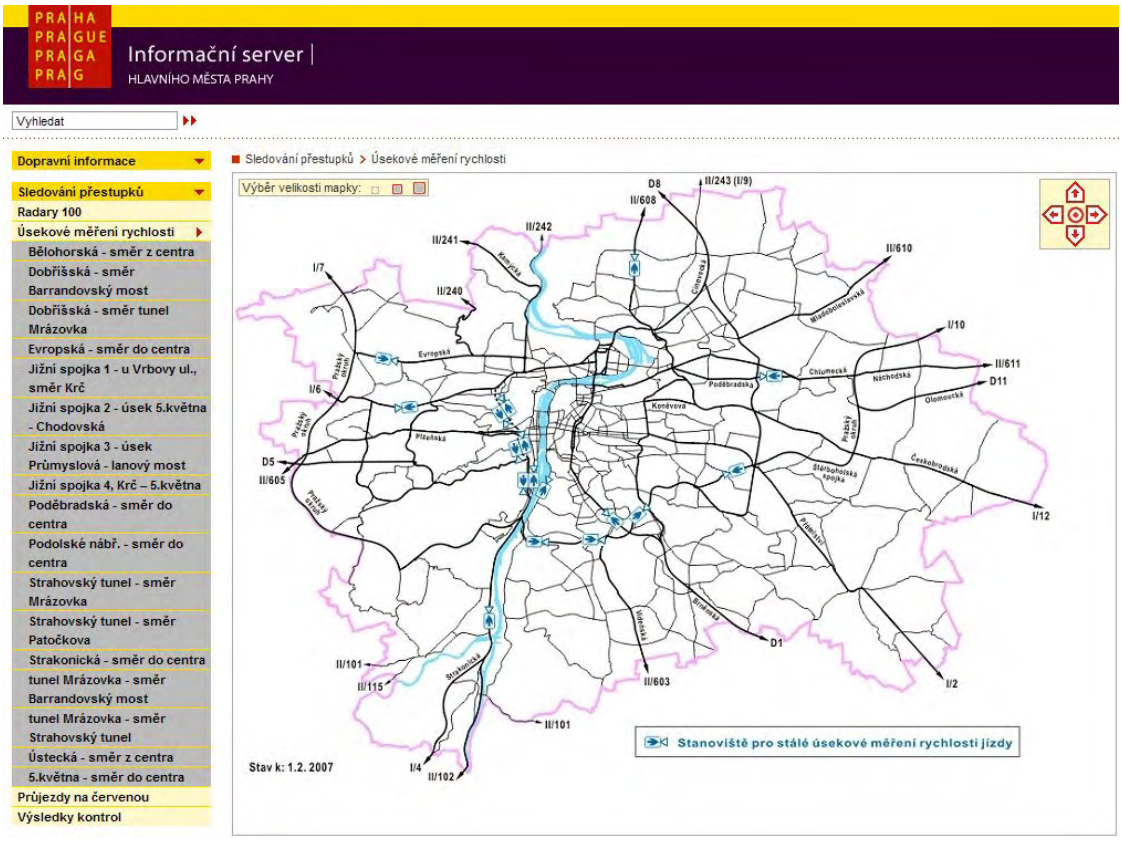
Pro potřeby stanovení doby průjezdu vozidla měřícím úsekem se jednoznačně určuje, že jak na vjezdu, tak na výjezdu z měřicího úseku bylo měřeno stejné vozidlo. Vozidlo se porovnává na základě registrační značky SPZ/RZ1 resp. SPZ/RZ2 pořízené v referenčních místech s_1 resp. s_2. Uvedený test se nazývá ztotožněním a je realizován opět pomocí algoritmů počítačového vidění a umělé inteligence. Ztotožnění se provádí se všemi referenčními snímky pořízenými v referenčním místě s_1 s referenčními snímky z místa s_2. Ztotožnění se provádí též v případě, že je rychloměr instalován na více než jednom jízdním pruhu, kdy je třeba křížově kontrolovat RZ

všech vozidel na výjezdu s vozidly na vjezdu do měřícího úseku. Platí, že pokud řidič přejede z jednoho jízdního pruhu do druhého, bude mu vždy naměřena průměrná rychlost nižší, než kterou ve skutečnosti jel a namůže tedy být poškozen.

Dokladem o přestupku překročení maximální povolené rychlosti jsou dva snímky, pokud je z nich zřejmé, že naměřená rychlost je vyšší než povolená, doplněny o údaje potřebné k prokázání přestupku. Přestupkové dokumenty se archivují na záznamové médium rychloměru, odkud si je odebírá uživatel. Přestupkové dokumenty jsou dále, při tzv. přestupkovém řízení, kontrolovány školeným operátorem na PC pomocí vyhodnocovacího programu.

Snímky na vjezdu a na výjezdu z měřícího úseku obsahují tyto údaje: časové razítko, identifikace a název místa měření, délka měřicího úseku, doba průjezdu, pořadové číslo dokumentu, výrobní číslo rychloměru, aktuálně nastavený limit maximální povolené rychlosti a naměřená minimální průměrná rychlost vozidla. Přesnost měření[16] je zaručena tím, že vzdálenost měřicích míst je velmi přesně (geodeticky) zaměřena a oba snímky z detekčních oblastí (řezů) jsou opatřeny přesnými časovými razítky ze stabilní časové základny synchronizované z GPS s přesností na 1 ms vzhledem k univerzálnímu času (GMT).

V ČR se v současnosti pomocí technologie Unicam měří úseková rychlost na více než 30 úsecích (více než 120 kamer). Např. v Praze je jejich umístění předem zveřejněno viz obrazovka na Obr. 40 serveru hl. m. Prahy, na které jsou uvedena nejen místa, kde jsou rychloměry instalovány, ale též aktuální statistiky o poklesu přestupků.



Obr. 40 Umístění systémů MÚR Praze

Zdroj: Informační server hl. m. Prahy

[16] Synchronizace systému na čas systému GPS s absolutní přesností na 1ms je unikátní vlastností systému Unicam.

12.3 DETEKCE JÍZDY NA ČERVENOU

Dalším dopravním vizuálním systémem je zařízení pro detekci jízdy na červenou (DJČ) a je na trhu v ČR již od roku 1996. Jedná se o původní český výrobek (obchodní název UnicamREDLIGHT) vyvinutý a vyráběný firmou CAMEA.

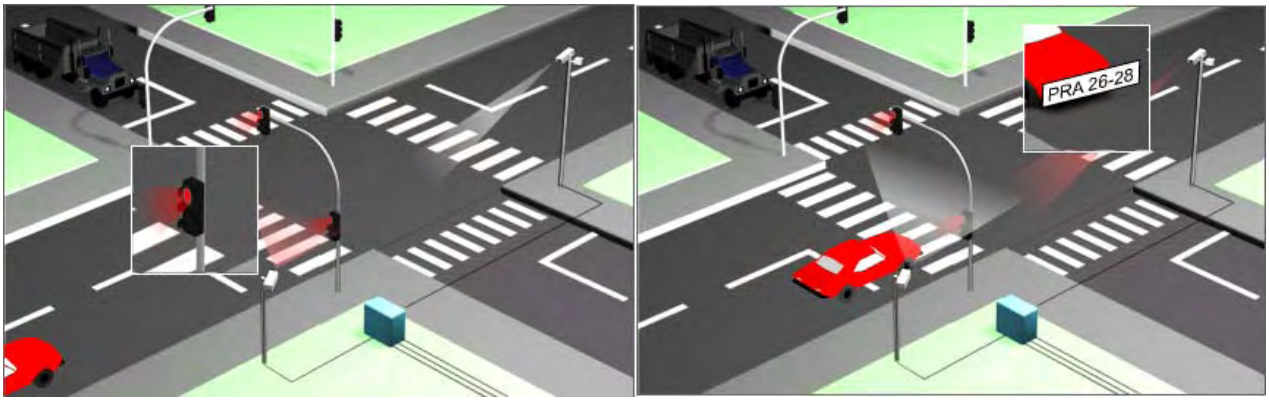
Hlavní inovací je využití kamer a videodetekce jak pro rozpoznání fáze signalizačního zařízení, tak rozpoznání vjezdu vozidla do křižovatky (přejezd stop-čáry). Klasické systémy pro DJČ jsou napojeny na řadič signalizace (pro zjištění fáze semaforu) a pro detekci vozidla v křižovatce využívají převážně indukčních smyček zařezaných do vozovky. Od první instalace v roce 1995 prošel systém pro DJČ řadou inovačních změn. Současné řešení představuje již třetí generaci zohledňující jak technologický pokrok, tak především zkušenosti s jeho použitím policií a správními orgány. Na základě potřeb uživatelů byla v průběhu let zapracována řada nových funkcí a vlastností. Výsledkem vývoje je produkt, který byl vyvinut na „míru“ potřebám uživatele.

Popis

Pořízení přestupku je plně automatizovaná činnost, což eliminuje vliv lidského faktoru. Zdokumentované přestupky jsou jednoznačně identifikovány místem, datem, hodinou jeho spáchání. Okamžik spáchání přestupku je určen s vysokou přesností a stabilitou. Dále jsou přestupky elektronicky podepsány a při přenosu šifrovány a nelze je tedy modifikovat či zneužít.

Použití videodetekce pro rozpoznání situace na křižovatce pro potřeby DJČ sebou nese řadu úskalí: jednoduché řešení, kdy se videodetekčnímu algoritmu v obraze vyznačí poloha semaforu, v praxi nefunguje. Pokud se chvějí sloupy, na kterých je kamera či semafor umístěn, dochází k nesprávnému rozpoznání fáze semaforu. Řešením je automatické dohledávání semaforu, což je však velmi složitá úloha. Systém DJČ má tento problém velmi robustně vyřešen díky dlouholetým zkušenostem s rozpoznáváním fáze desítek různých semaforů na řadě křižovatek. Kamery systému automaticky detekují vozidlo v křižovatce pomocí videodetekce. Dále sledují jeho pohyb s cílem eliminace bočních vjezdů a zastavení v křižovatce. Výsledkem je vyloučení sporných přestupků.

V základní sestavě je na křižovatce umístěna jedna přehledová kamera pro 1 až 3 jízdní pruhy a jedna detailová kamera pro každý měřený pruh vozovky. Pomocí přehledové kamery, která je umístěna po směru jízdy vozidla, se zjišťuje fáze semaforu. Dále se zaznamenává celková situace na křižovatce, historie vjezdu vozidla do křižovatky a pořizuje se digitální videozáznam celkové situace, např. pro potřeby prokazování dopravních nehod Detailovými kamerami umístěnými v protisměru se zjišťuje vozidlo v prostoru křižovatky zepředu s cílem pořízení detailního snímku vozidla včetně čitelné registrační značky v místě stop-čáry a tváře řidiče v celé šířce jízdního pruhu.



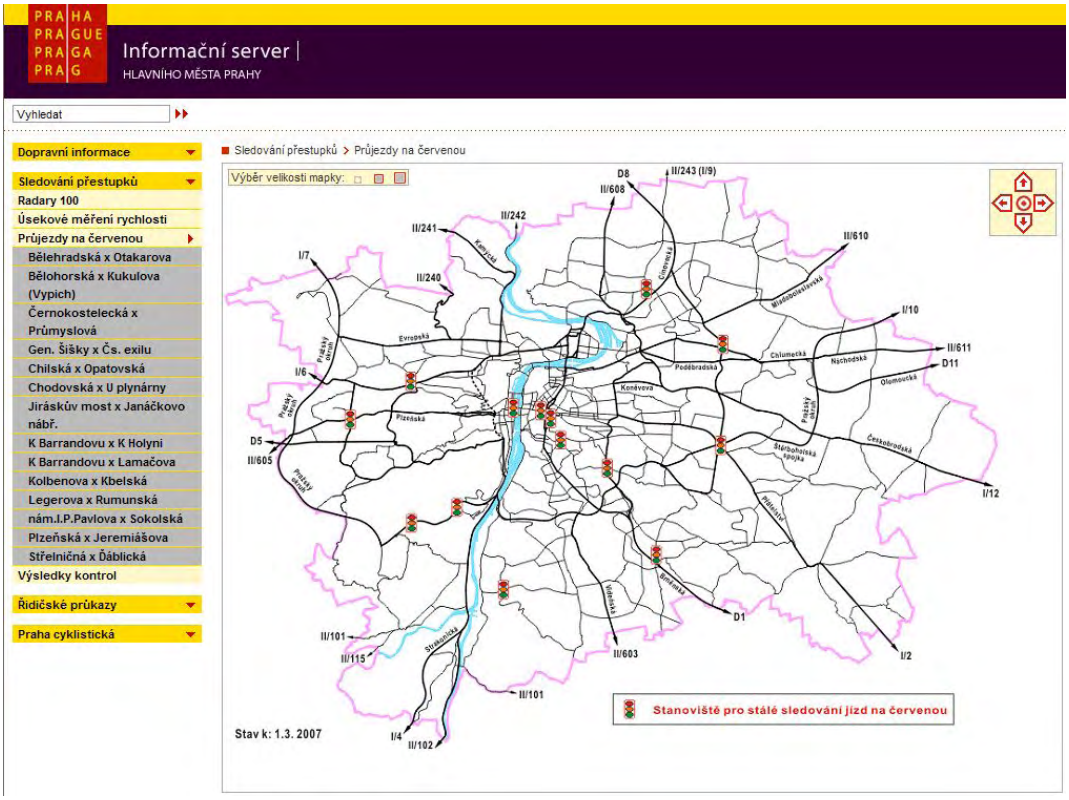
Obr. 41 Ilustrace činnosti systému DJČ

Vyhodnocovací server (počítač) zpracovává aktuální situaci detekovanou jednotlivými kamerami, uschovává údaje o přestupcích a zaznamenává obrazovou informaci situace na

křižovatce, např. pro potřeby vyšetřování případné dopravní nehody. Na Obr. 41 je znázorněna činnost při rozpoznání fáze semaforu a pořízení snímku vozidla po přejezdu stop-čáry.

Systém automaticky vyhledává a čte SPZ/RZ vozidel detekovaných v prostoru křižovatky. Pracuje v reálném čase a přečtená RZ je k dispozici bezprostředně po detekci vozidla. Úspěšnost správného čtení je velmi vysoká. Díky tomu se provádí primární třídění snímků na základě vyhodnocení RZ.

V ČR se v současnosti pomocí technologie Unicam detekuje jízda na červenou na více než 80 jízdních pruzích (více než 120 kamer). Např. v Praze je jejich umístění předem zveřejněno viz Obr. 42 obrazovka na serveru hl. m. Prahy, na které jsou uvedena nejen místa, kde jsou rychloměry instalovány, ale též aktuální statistiky o poklesu přestupků.



Obr. 42 Umístění systémů DJČ Praze

Zdroj: Informační server hl. m. Prahy

12.4 SBĚR DOPRAVNÍCH DAT

Vizuální systém Unicam zjišťuje též intenzitu silničního provozu, obsazenost virtuálních detektorů a skladbu dopravního proudu (typ vozidel, rychlost vozidel) pomocí videodetekce. Systém vykazuje robustní necitlivost na tzv. falešné detekce způsobené povětrnostními vlivy, jako je např. vliv slunce (mraky a stíny) atd. a je uzpůsoben k nepřetržitému provozu.

Zjišťované údaje

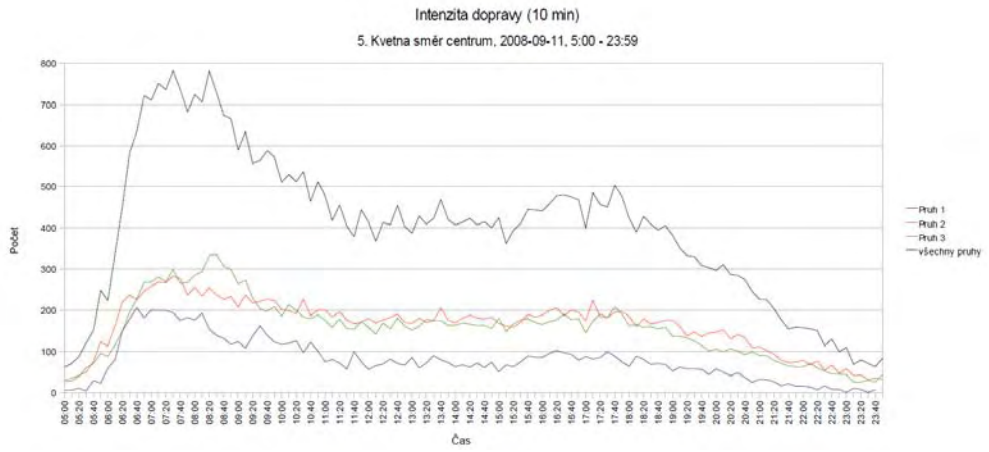
Kamery systému sledují vždy jedna jeden jízdní pruh. Systém detekuje průjezd vozidla v zorném poli kamery (tzv. virtuální detektor) a pomocí videodetekce zjišťuje následující parametry:

- Intenzita dopravy (hustota silničního provozu).
- Obsazenost virtuálních detektorů (jízdních pruhů).
- Skladba dopravního proudu vzhledem na rychlost projíždějících vozidel.

- Skladba dopravního proudu vzhledem na typ projíždějících vozidel.

Intenzita dopravy

Intenzita dopravy je určována jako počet vozidel, která projela daným profilem komunikace za daný čas. Počet vozidel je určován na základě algoritmů počítačového vidění. Výskyt pohyb v definované části obrazu, směr a velikost pohybu v definované části obrazu a viditelnost vozovky anebo přítomnost automobilu. Systém je konstruován tak aby byl schopen dodat relevantní informace i v případě nepříznivých povětrnostních situací jako šero, ostré protisvětlo, mokrá vozovka a ostré stíny.

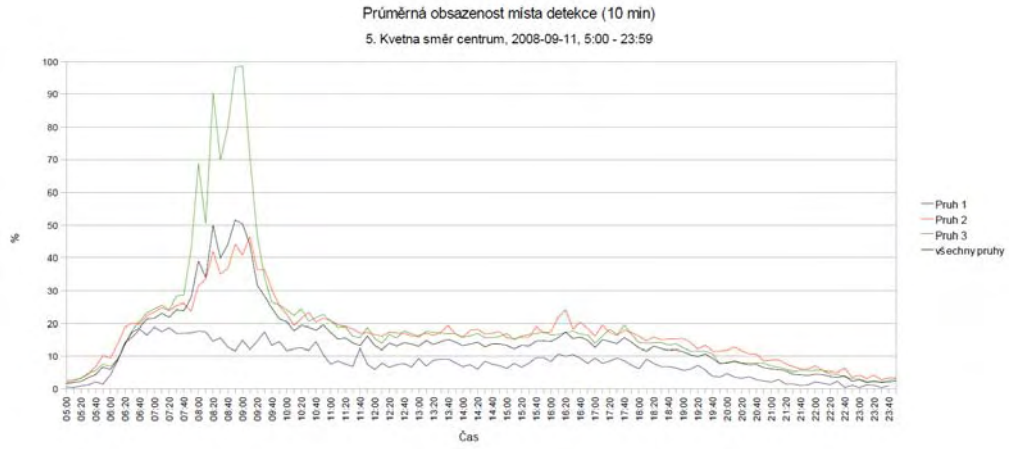


Obr. 43 Graf intenzity dopravy

Na Obr. 43 je znázorněn graf intenzity dopravy zjištěné předmětným kamerovým systémem. Na daném grafu jsou zobrazeny intenzity v 10 minutovém intervalu (časový interval lze nastavit dle potřeby) pro každý jízdní pruh zvlášť a současně pro celý jízdní pás

Obsazenost virtuálního detektoru

Jedná se tedy o podíl součtu jednotlivých dob, kdy je virtuální detektor obsazen konkrétním vozidlem či vozidly za daný časový interval k délce tohoto intervalu. I v tomto případě jsou použity algoritmy počítačového vidění. Na Obr.44 je znázorněn graf průměrné obsazenosti virtuálního detektoru zjištěné předmětným kamerovým systémem.

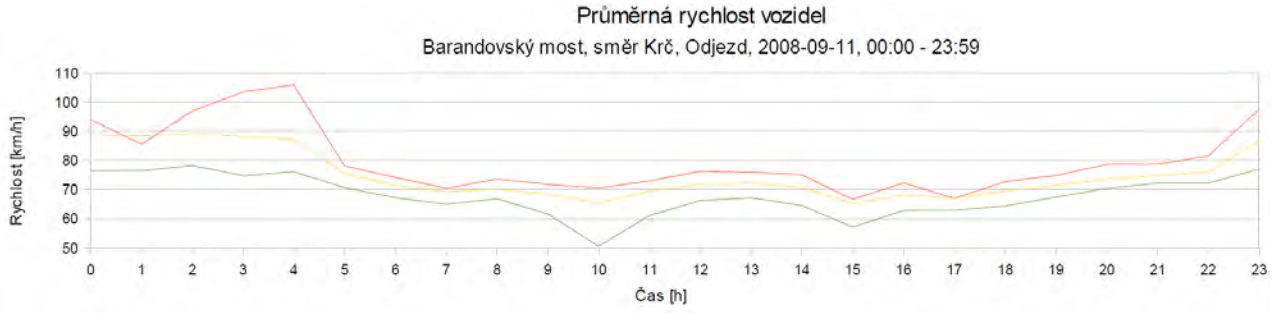


Obr.44 Graf obsazenosti virtuálního detektoru

Na daném grafu je zobrazen vývoj průměrné obsazenosti v průběhu dne pro každý jízdní pruh zvlášť a současně pro celý jízdní pás. Průměrná obsazenost je v tomto případě vypočítávána v rámci 10 minutových intervalů (časový interval lze nastavit dle potřeby).

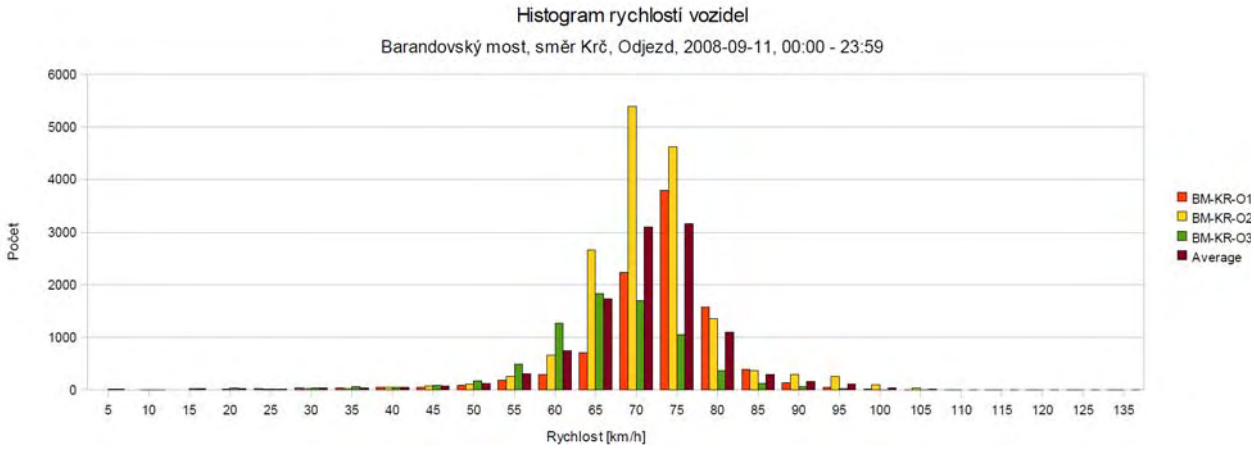
Skladba dopravního proudu vzhledem na rychlost projíždějících vozidel

Informace z úsekového dopravního detektoru lze agregovat a statisticky vyhodnocovat. Průměrná rychlost se tedy stanovuje jako podíl součtu rychlostí jednotlivých vozidel za daný časový interval, ku počtu všech vozidel za daný časový interval. Na Obr. 45 znázorněn graf vývoje průměrné rychlosti vozidel v průběhu dne.



Obr. 45 Průměrná rychlost vozidel v průběhu dne

Dále se jedná se o počty vozidel, která projížděla měřeným úsekem v jednotlivých rychlostních skupinách. Na Obr. 46 je znázorněn histogram rychlostí zjištěné předmětným kamerovým systémem. Na daném grafu jsou zobrazeny jednotlivé rychlostní kategorie za celý den.



Obr. 46 Histogram rychlostí vozidel

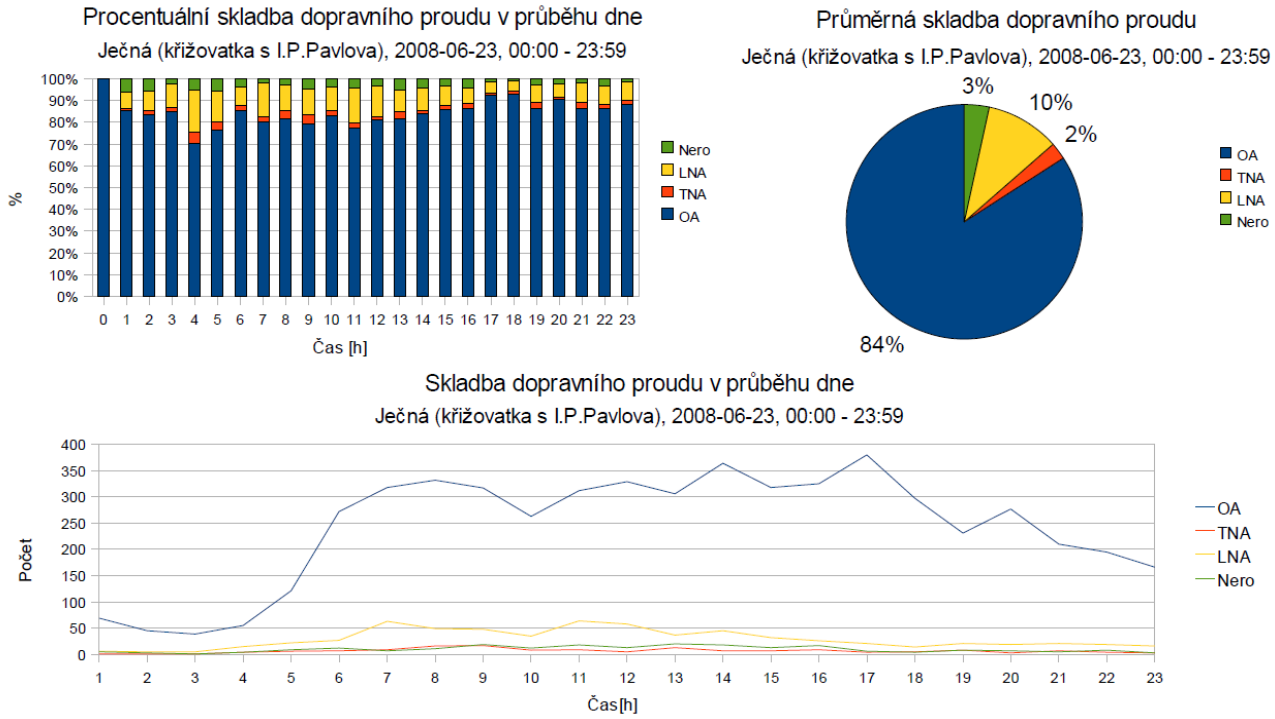
Skladba dopravního proudu vzhledem na typ projíždějících vozidel

Automobily jsou zařazeny do následujících přednastavených kategorií:

- OA osobní vozidla,
- LNA nákladní vozidla lehká
- TNA nákladní vozidla těžká
- Nero neurčená vozidla

Pro každou z uvedených kategorií je měřen počet vozidel za daný časový interval. Jsou použity algoritmy počítačového vidění a umělé inteligence. Rozpoznávány jsou jak denní tak noční snímky (použito je noční přisvětlení masky). Na Obr. 47 jsou znázorněné následující grafy:

Procentuální skladba dopravního proudu v průběhu dne, průměrná skladba dopravního proudu a skladba dopravního proudu v průběhu dne.



Obr. 47 Skladba dopravního proudu

12.5 TECHNOLOGE UNICAM

V tomto odstavci je uveden stručný přehled klíčových technologických komponent, ze kterých se sestavují konkrétní aplikace systému Unicam.

Noční vidění

Obr. 48 Snímky vozidla v noci

Systém Unicam je vybaven systémem nočního vidění. Díky osvětlovacím jednotkám pořizuje systém ostré, nerozmazané snímky vozidla jedoucího rychlostí 156,4 km·h^{-1}, na kterých je vidět jak RZ, tak typ vozidla a tvář řidiče. Jedná se o speciální infračervené (IR) reflektory a blesky,

které umožňují pořízení snímků vozidel za tmy a snížených světelných podmínek včetně masky vozidla. Infračervené osvětlovací jednotky svítí na projíždějící vozidla zepředu tak, aby byly viditelné jejich RZ, infračervené blesky pak osvítí celé vozidlo. Realizace systému nočního vidění je řešena podle užitného vzoru „Zařízení pro monitorování vozidel elektronickým kamerovým systémem“ drženého společností CAMEA, spol. s r.o.

Inteligentní kamery s vysokým rozlišením

Kamery systému Unicam mají vysoké rozlišení a vestavěnou inteligenci, která umožňuje, aby kamera sama detekovala vozidlo bez nutnosti externích čidel. Díky tomu lze kamery jednoduše instalovat (popis je též v odstavci 4.8).

Obr. 49 Intalace systému Unicam v městské zástavbě

Konstrukce a instalace

Obr. 49 ilustruje provedení a vzhled kamerového systému instalovaného v městské aglomeraci. Kamery lze instalovat např. na sloupy veřejného osvětlení, trakční sloupy apod. na straně vozovky. Z principu činnosti vyplývá, že systém je neinvazivní vůči vozovce.

Systém Unicam je optimalizován pro nízký příkon. Díky tomu tam, kde není k dispozici pevná elektrická přípojka, mohou být kamery a výpočetní jednotky v noci napájeny ze sloupů veřejného osvětlení a ve dne z baterií. Dle dostupných informací není na trhu žádné zařízení podobného druhu, které by bylo možno napájet obdobných způsobem.

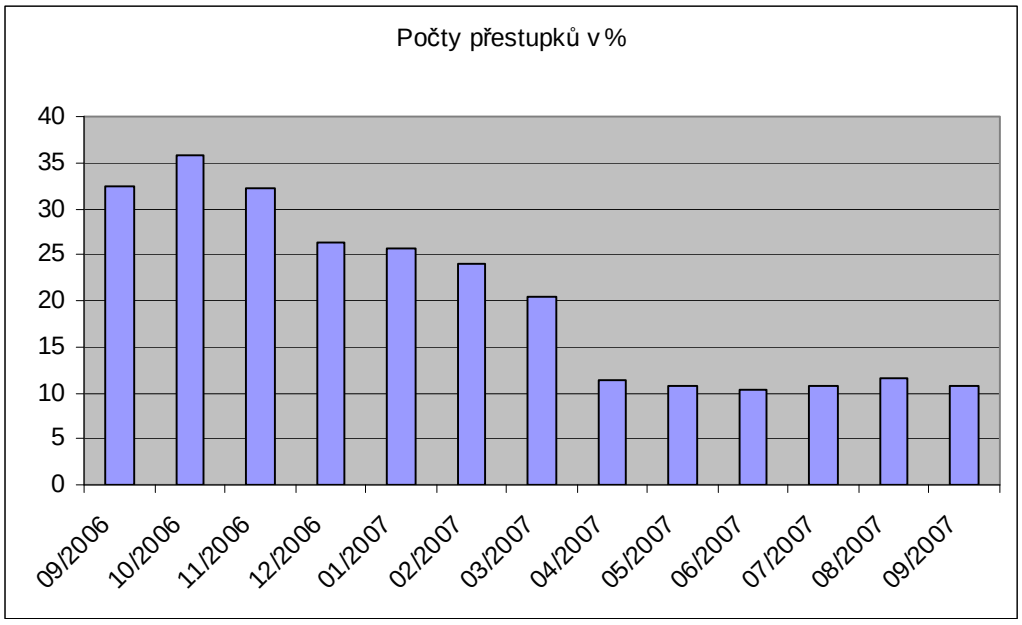
Veškerá datová komunikace mezi kamerami a centrálním serverem může probíhat i bezdrátově. Jedním z důvodů nízkých nároků na přenosové kapacity je i to, že kamery mají vestavěnu inteligenci, díky které se posílají k dalšímu zpracování jen ty snímky, na kterých je s velkou pravděpodobností vozidlo. Eliminace kabelové infrastruktury jednak snižuje cenu instalace zařízení a dále pak umožňuje kamery instalovat bez nutnosti výkopů či převěsů a umožňuje montovat kamery prakticky kamkoliv (např. sloupy veřejného osvětlení, portály, apod.). Kamerový systém tak lze instalovat bez stavebního povolení a dodatečných stavebních nákladů. Díky bezdrátové komunikaci lze veškerá naměřená data a přestupky přenášet do ústředny on-line. Protože jsou přestupky k dispozici ihned po jejich spáchání, lze provádět též on-line lustrace.

12.6 SHRNUTÍ

ITS technologie prochází prudkým rozvojem a je zřejmé, že inovativní řešení, s vyžitím nových technologií, mají prokazatelný přínos pro zlepšení životních podmínek obyvatel díky adresnému řešení problémů dopravy. Představují též velmi atraktivní komoditu na trhu se značným potenciálem.

ITS systém, který by nabízel takový stupeň multifikčnosti, není dle dostupných informací nikde v ČR provozu a představuje nejkomplexnější inteligentní kamerový systém svého druhu v ČR. Veškeré klíčové technologie a komponenty systému jsou výsledkem původního vývoje (Camea). V průběhu let byly též získány rozsáhlé zkušenosti s výzkumem, vývojem, výrobou, instalací a údržbu rozsáhlých multifinkčních kamerových systémů.

Na základě zkušeností s jeho provozem lze konstatovat, že jeho funkce, stupeň integrace a globální zpracování dopravních údajů představuje výrazný přínos pro zvýšení bezpečnosti silničního provozu, snížení emisí a hluku a k lepšímu řízení a plánování dopravy. Uvedené přínosy představují výrazné společenské úspory, které lze jen obtížně ekonomicky vyčíslit. Pro ilustraci velkých a pozitivních společenských dopadů lze uvést, že např. v roce 2007 byl v hl. m. Prze zaznamenán meziroční pokles dopravních nehod s fatálními následky o 40 %, (viz ohlasy v médiích [4]). Uvedených výsledků bylo dosaženo z velké části díky aktivnímu nasazení silničních rychloměrů technologie Unicam. Na Obr. 50 je uveden histogram počtu přestupků překročená rychlosti, který výše uvedené závěry dokumentuje. Zdrojem dat je Městská policie Praha a Informační server hl. m. Prahy.



Obr. 50 Pokles počtu přestupků překročení rychlosti v hl. m. Praze

12.7 ODEZVY V MÉDIÍCH

[1] Wikipedia. otevřená encyklopedie [online]. Řízení dopravy (Doprava v Praze). [cit. 2008-12-08]. Dostupné z WWW: < http://cs.wikipedia.org/wiki/Doprava_v_Praze >.

[2] Kamerový systém je účinný. [cit. 2008-12-08]. Dostupné z WWW: http://www.praha.eu/jnp/cz/extra/metamorfozy/mestska_policie/kamerovy_system_je_ucinny.html.

[3] Bič na zloděje aut. [cit. 2008-12-09]. Dostupné z WWW: 2006\Kamerový systém je účinný, poroste už pomaleji 20080615.pdf.

[4] Tisková zpráva. [cit. 2008-12-11]. Dostupné z WWW: http://magistrat.praha-mesto.cz/77004_Prohlaseni-prvniho-namestka-primatora-Rudolfa-Blazka.

[5] Doprava v hl. m. Praze. [cit. 2008-12-11]. Dostupné z WWW: http://doprava.praha-mesto.cz.

[6] Kamery pracují za policisty. Lidové noviny. [cit.2007-03-07]. Dostupné z WWW: 2006\Kamery měří rychlost - Centrum_cz Motožurnál.htm.

[7] Poměrové měření rychlosti. [Cit.2007-04-02]. Dostupné z WWW: http://auto.idnes.cz/pomerove-mereni-rychlosti-jak-to-funguje-f11-/automoto.asp?c=A050421_075455_ak_aktual_cap.

[8] Pražští strážníci mají nové radary. [cit. 2007-04-08]. Dostupné z WWW: http://auto.idnes.cz/prazsti-straznici-maji-nove-radary-duh-/automoto.asp?c=A061019_203706_automoto_fdv.

[9] Kde číhá policie. Idnes.cz. [cit.29.11.2006]. Dostupné z WWW: http://auto.idnes.cz/kde-ciha-policie-kompletni-prehled-radaru-v-praze-fbz-/automoto.asp?c=A061128_172505_automoto_fdv.

[10] Zlínské řidiče hlídají kamery. TV Nova. [cit. 14. 5. 2007]. Dostupné z WWW: http://www.nova.cz/zpravy/?83c=%7Edomaci%7E&83e=DO3300&ex3300=zlinske-ridice-hlidaji-kamery.

[11] Zlínské řidiče hlídají kamery. Zlínské noviny, 2007-01-08.

[12] Neukázněné řidiče sleduje kamerový systém. Právo, 2007-03-21.

[13] Chytrá kamera už vydělala miliony. Dnes, 2007-03-22.

[14] Špičkový radar je v ohrožení. Lidové noviny, 2007-09-03.

[15] Doprava je klidnější, ale Brno zaostává. Rovnost, 2007-03-20.

[16] TV Markíza, večerní zpravodajství, 2008-07-01.

[17] ČT1, Události v regionech, 2007-12-14.

[18] Dopravních kamer přibude. MF Dnes, 2009-03-19.

[19] Radary snižují rychlost i hluk. Právo, 2008-05-15.

13 PŘÍLOHA 2 – PUBLIKACE CHARAKTERIZUJÍCÍ INŽENÝRSKÁ DÍLA

Pozn.: Tato habilitační práce je podána formou souboru významných inženýrských děl s velkým společenským dopadem a ohlasy, který je doplněn komentářem. S ohledem na značný rozsah dokumentace k jednotlivým systémům, jsou jejich hlavní charakteristiky prezentovány ve formě následujících publikací[17]:

1. FUČÍK, O.; ZEMČÍK, P.; TUPEC, P.; CRHA, L.; HEROUT, A. The Networked Photo-Enforcement and Traffic Monitoring System. In: *Proceedings of Engineering of Computer-Based Systems, ECBS 2004.* Los Alamitos, USA: 2004, pp. 423-428. ISBN 0-7695-2125-8.
2. ZEMČÍK, P.; HEROUT, A.; CRHA, L.; FUČÍK, O.; TUPEC, P. Particle rendering engine in DSP and FPGA. *In Proceedings of Engineering of Computer-Based Systems, ECBS 2004.* 2004, pp. 361- 368. ISBN: 0-7695-2125-8.
3. ZEMČÍK, P.; FUČÍK, O.; HONEC, J.; VALENTA, P. Visual Analysis on the Production Line. In *Proceedings of the 10th Scandinavian Conference on Image Analyses*. Lappeenranta, Finland: 1997, pp. 308-310.
4. NOVOTNÝ, J.; ANTOŠ, D.; FUČÍK, O. Project of IPv6 Router with FPGA Hardware Accelerator. In *Proceedings of the Field-Programmable Logic and Applications, FPL 2003.* Edit. by Cheung P.Y.K.; Constantinides G.A.; de Souza J.T. Berlin-Heidelberg: Springer, Lecture Notes in Computer Science 2778, 2003, pp. 964-967. ISBN 3-540-40822-3.
5. ZEMČÍK, P.; HEROUT, A.; BERAN, V.; FUČÍK, O.; SCHIER, J. Reconfigurable Image Processing Architecture. International Journal on Graphics, Vision and Image Processing (GVIP). Special Issue on Applicable Image Processing. 2006, pp.7-12. ISSN1687-3998.
6. ZEMČÍK, P.; HEROUT, A.; BERAN, V.; POTÚČEK, I.; FUČÍK, O.; HONEC, J.; RICHTER, M; KALOVÁ, I.; LISZTWAN, M. Image Processing in Traffic Applications. International Journal on Graphics, Vision and Image Processing (GVIP). Special Issue on Applicable Image Processing. 2006, pp.7-12. ISSN1687-3998.
7. BRYAN, L.; FUČÍK, O. FPGA Implementation of a Reconfigurable License Plate Detection Method. In *Proceedings of the Engineering of Reconfigurable Systems and Algorithms, ERSA 2007*. Las Vegas, USA: 2007, pp. 1-4. ISBN 1-60132-026-4.
8. CRHA, L.; FUČÍK, O.; ŠUSTEK, J. Environment for Hw/Sw Codesign of Embedded Systems, In *Proc. of 8th IEEE Design and Diagnostic of Electronic Circuits and Systems Workshop*, Sopron, HU: 2005, s. 236-240. ISBN 9639364487.

[17] Podrobnou dokumentaci k jednotlivým inženýrským dílům je možno vyžádat u autora.

The Networked Photo-Enforcement and Traffic Monitoring System Unicam

Otto Fučík, Pavel Zemčík, Pavel Tupec, Luděk Crha, Adam Herout
Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic
[fucik, zemcik, tupec, crha, herout]@fit.vutbr.cz

Abstract

The presented system (Unicam) offers a complex state-of-the-art machine vision equipment and technology to provide automated video image vehicle detection devices dedicated for traffic monitoring applications. The system provides real time video image capturing, digital signal processing, compression, storage, and transmission over communication interfaces. It uses proprietary artificial intelligence algorithms and special image processing modules to achieve highly accurate vehicles detection. According to the users' needs, the system can be used for detection of red-light violations at road intersections, speed measurement, traffic data collection, video recording, or surveillance. Yet another possible application of the system is surveys based on license plate recognition for transportation engineers, stolen car searching, or toll-tag data collection. The system functionality has been improved by coupling camera sensors with specialized real-time processing units and adding networking capability. Implementation of video detection algorithms, hardware design units, and networking features are also discussed.

1. Introduction

The Unicam system [1] is based on video-detection devices that are using a microcomputer to analyze video images acquired by a video camera. Two basic techniques, tripline and tracking, are used for traffic analysis. Tripline techniques monitor specific zones in the video image to detect the presence of a vehicle. Video tracking techniques employ machine vision algorithms to identify and track vehicles as they pass through the field of view. Video technology can offer a wide variety of traffic information. In addition to conventional data such as volume, presence, occupancy, density, speed and classification, other data such as dwell time, traffic accident detection, and even origin/destination information can be obtained. Video can also be used to provide surveillance information on a roadway as well as photo-enforcement of traffic violations like red-light running and speeding.

Figure 1. Unicam sites in Prague

The Unicam system offers advanced machine vision technology to provide automated video image vehicle detection. It uses proprietary algorithms and special image processing modules to achieve high accuracy while keeping the overall system's price low. The system is in use since 1996 at several sites in the Czech

Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04)
0-7695-2125-8/04 $ 20.00 © 2004 IEEE

Republic. For example, more than 100 cameras are installed at different locations in Prague - see Fig. 1

Some of the possible applications of the Unicam system:

UnicamCross – records red-light violating vehicles. This subsystem continuously monitors the lights at an intersection. The basic principle of this application is shown in Fig. 2: The camera itself is triggered and the image is taken in the case when (1) a specified time elapsed after the signal has turned red, and (2) a vehicle is passing over the stop-line. Other images that typically show the red-light-violator moving in the intersection are also taken.

UnicamVelocity – records speeding vehicles, measures the average speed of a vehicle measured in the relatively long distance between two of more installed cameras – Fig. 3.

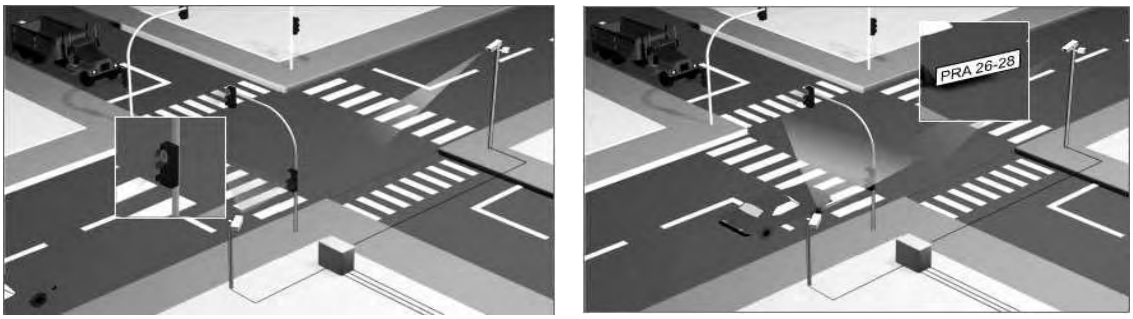


Figure 2: Red-light violation detection principle

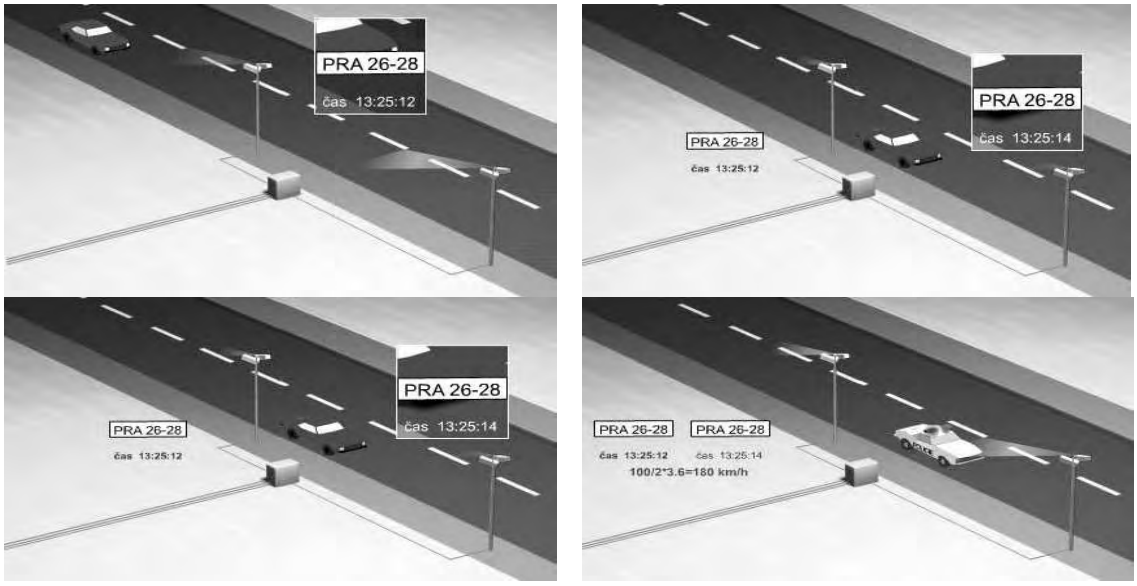


Figure 3: Average velocity measurement principle

UnicamLPR – recognizes the number plates of vehicles. This unit uses sophisticated optical character recognition (OCR) technology and is mostly used as a part of other subsystems, e.g. UnicamVelocity.

UnicamScan – for traffic flow monitoring. This unit is watching passing traffic and monitoring where it is going on the public road network. It can be used to measure the current traffic flow speed, volume, the speed of individual vehicles, and to predict likely trouble spots further ahead.

2. Novel approach – UnicamNet

The goal of the Unicam developers was also to add networking capability to the system to identify traffic law violators and monitoring traffic fluency on-line. Such networked system would have wide range of use and emerging properties.

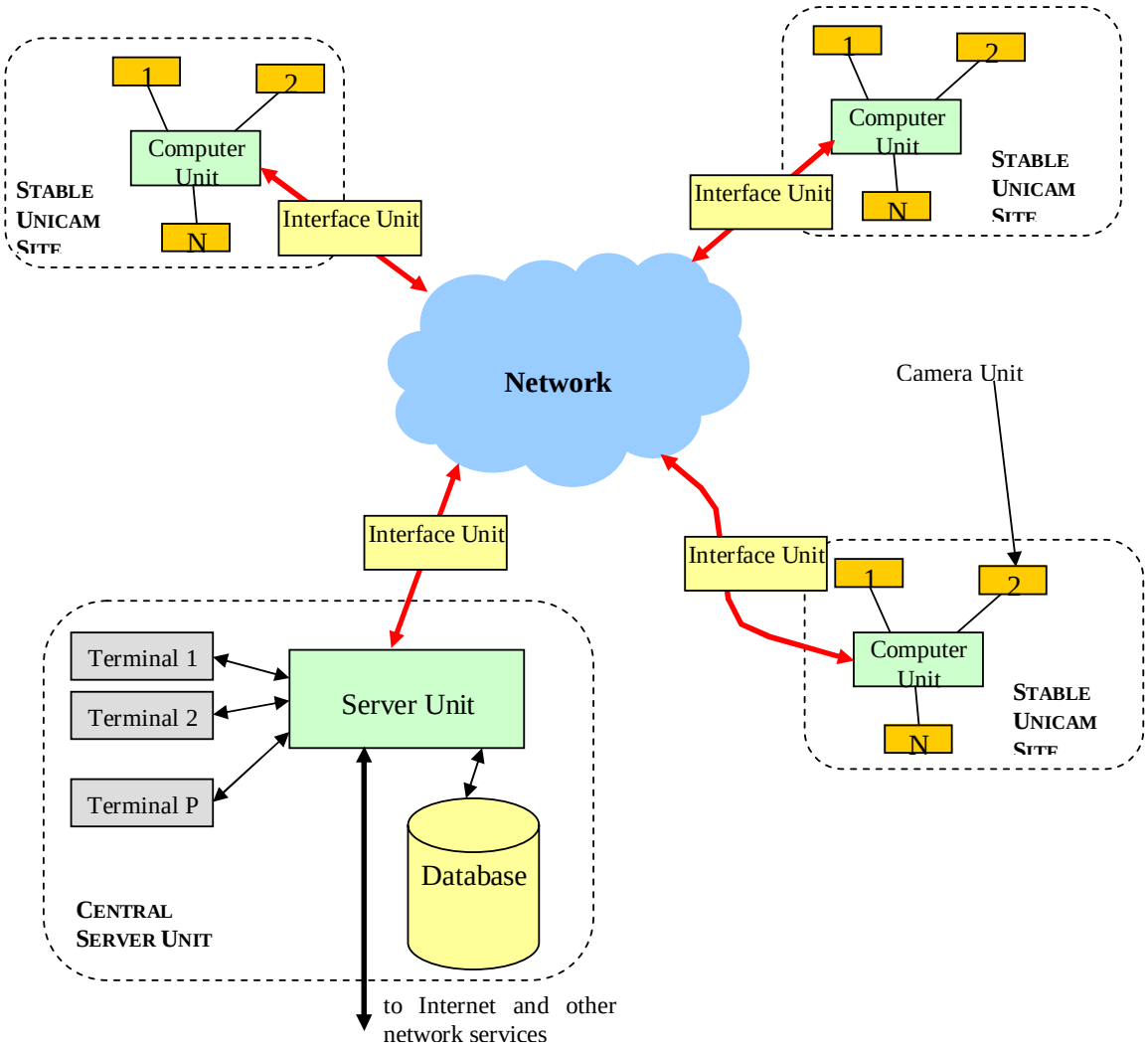


Figure 4: UnicamNet block diagram

The final project's significant impact on increased traffic safety and fluency will hopefully lead to its establishment also on the European market. To be able to fulfill such goal there are many steps that should be done, for example: To develop an intelligent video device with a processing element closely-coupled with the camera sensor and analyzing the video in real-time. In such device, the video sequence is being processed and the output of the intelligent device can be just the result – e.g. recognized license plates of vehicles. It allows for adding yet another capability to the Unicam system – networking. Rather than transfer real video sequences over the network, the system components transmit only the processed data. This significantly reduces the amount of data as well as speeds up its exchange.

The block chart describing the whole UnicamNet system is in Fig. 4. Two basic network components of the UnicamNet system are available: Stable Unicam site and Mobile Unicam Site. Each of them has a processing unit with peripheral camera units. Both the Stable Unicam Site and Mobile Unicam Site are connected to the network through interface units. The server unit containing the central computer and several terminals are connected to the network through the same interface unit enabling several users to work with the system. The gained data and statistics are stored in a database. The server unit also includes an Internet interface and other network services.

In the following paragraphs, selected parts of the intelligent video device (IVD) are described – a camera

unit closely coupled with real-time video processing unit and a high speed communication interface.

3. Innovation

The embedded processing solution can be more efficient than the traditional one. It is more compact, better serviceable, etc., but in fact, the main advantage is that the system is distributable. This means that the cameras can be placed remotely from the PC that integrates the system functionality, and that the connection of the camera units can be done through the network connection which is easier to implement than proprietary analog or digital video signal connection. In addition, this approach allows for creation of widely distributed systems running in the agglomerations with very long distances between the intelligent cameras and PCs. The basic differences between the traditional computer based approach and the embedded processing approach is shown in Fig. 5.

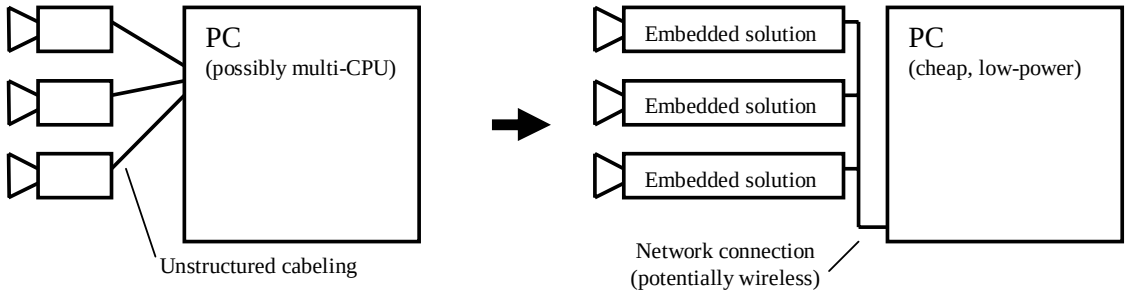


Figure 5: Traditional and embedded approach

4. Embedded software

The embedded application software runs under a small operating system (OS) on a DSP processor by Texas Instruments (TI). The OS is based on DSP/BIOS supplied by TI for the selected DSP processor. The main idea of the application software development is to have identical implementations on both computer (PC) and embedded DSP. This feature allows developing some non real-time parts of applications in personal computer environment and their later compilation for the DSP. The advantage is that the PC environment offers more comfortable debugging and testing the program.

The kernel has been developed in order to offer many features comparable to commonly used operating systems. All running threads are sorted into classes depending on priority and their current state (running, waiting/stopped, ready to go). For the rest of the interrupt operations Deferred Procedure Call is called from the ISR later. These callback functions have high priority in a front of the waiting threads and they are executed before the others common threads. This concept ensures fast reactions of the systems while accessing the close neighbors. Functions dealing with Direct Memory Access (DMA) and functions working with L1 and L2 cache coherency are also included.

Intelligent cameras communicate with remote units by Ethernet with variable bit-rate 10/100 Mb/s. Special module supporting the TCP/IP operations is implemented for DSP too. Module interface is similar to the BSDSockets used in UNIX operating systems. Fast 1Mb RS-232 serial link is used for some system operations and debug messages.

There is a special unit for treating the HDTV camera images. Data transfer is done in the background using DMA transfers. Images are stored into a round buffer serially. The buffer arbiter decides if any image should be discarded (images without cars or old ones). The order of image processing operations is determined by the arbiter in the following way: Every new incoming image passes thought a test which shows probability of occurrence of a vehicle with license plate in the image. If the probability of the license plate occurrence is high, then it is send to the next processing stage where the license plate is recognized (read).

5. Programmable hardware

There are two basic units that serve as a programmable hardware. While CPLD (because of its low speed and density) is used only as a boot manager and a device driver (Flash ROM, SMAP, UART,...), FPGA contains much more sophisticated units. In the following paragraphs basic hardware processing units are described.

5.1 Boot manager

It is placed in the CPLD. After reset, data starting at certain address in ROM are written to SMAP port, which causes loading the FPGA design.

5.2 Inter chip communicating subsystem

This system allows communication between every two devices on the board also with support of dynamical reconfiguration of the FPGA on-board chip. The system can be controlled and dynamically reconfigured by any of the devices connected.
In the Fig. 6 the device called the Core is a master of all the communication block diagram is shown. Device that wants to communicate with some other device asks the Core for transfer of the data. Detailed description of this system is in [2].

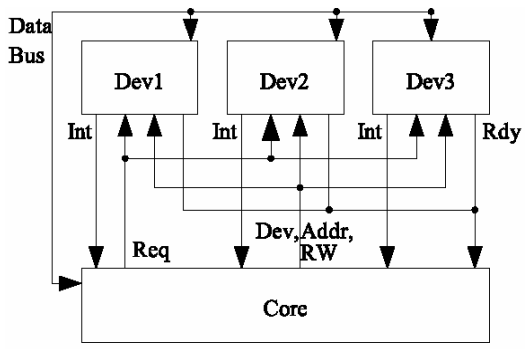


Figure 6: Inter chip communication core

5.3 Ethernet driver

This unit is a low-level driver for the Ethernet communications. It prepares data for the following processing in the DSP.
At the input, data are deserialized, CRC is checked and the data are put to the DSP through the buffer. At the output, data got from the DSP are buffered and sent out. There is also the MII driver implemented for allowing the DSP to read or write from/to the phyter circuit.

5.4 Grabber driver

There is a grabber on the board connected to an HDTV camera. It consists of the A/D converter for the data from the camera, D/A converter for setting up the gain and offset properties, and other related circuits. Driver consists of several units. A genlock generates the pixel clock for the grabber and also internal pixel clock to be used in FPGA. Main grabber unit counts the number of pixels passed on a row and rows passed in an image. These numbers are used for trimming the image and for setting the clamp signal. Data from the A/D converter are buffered and sent to the DSP for next processing. For this transfer, DMA is used. DSP can set important features by writing to internal FPGA registers, like value of the gain and offset, start and end of the clamp and the size and position of the image transferred to the DSP.

6. Communication and networking

Communication plays very important role in any computer application. This fact is even more emphasized in distributed systems such as the Unicam system. The communication can be, from the point of view of developer, subdivided into two categories (besides the known ISO/OSI and other known and standardized methods). The categories are:

- Communication inside the system
- Communication outside the system

While the communication outside the system is very desirable to be done in standardized way to allow interoperability with other systems, the communication inside the system does not necessarily have to be done that way and in some cases, e.g. efficiency can be of more importance than standardization. The communication inside the system can be further divided into several categories:

- Communication between the threads/processes
- Communication with peripheral devices and parts of the board(s)
- Communication between the boards

While the first two categories are critical from the point of view of speed, efficiency, and simple implementation, the latter one may be needed to implement in a standard way to allow commercially available communication parts on the path between the boards. Still, the developer of application most probably does not want to see the differences in the communication categories. This leads to the following design consideration:

- Implement a standard communication method for inter-board communication and communication with the outside world
- Implement as many as necessary proprietary communication subsystems as needed for efficient communication inside the board and inter-process
- Create an abstraction (software) to cover the differences between the categories and to allow

developer not to be aware of the category; the channels should ensure reliable connection with timeout and their software interface should be derived to some extent from the BSD-Sockets

At the moment, the intra-board communication is done by the means of fast shared memory, for the local inter-board links, RS-232 serial or LVDS lines and potentially the PCI bus can be used, for the wide range communication such as between distant smart cameras and the central computing system or between computer units engaged in the networked system, the BSD sockets (TCP/IP) are employed.

Since the IP protocol is used, the system may be accessible over all networks capable of IP datagram transfer. Thus, huge number of already existing and well debugged SW tools can be used for:

- Network monitoring
- Network traffic and reliability analysis
- Remote diagnostics
- Data encryption and authentication

A significant progress has been achieved in wireless communications in recent years. We are able to access the smart camera based systems through such networks, because most public wireless networks (namely GPRS in GSM networks) support the IP protocol.

7. Conclusions

In this paper there is described the field proven system Unicam which is based on a video-detection devices. The system is based on video tracking techniques that employ algorithms to identify and track vehicles as they pass through the field of view. Video technology can offer a wide variety of traffic information. In addition to conventional data such as volume, presence, occupancy, density, speed and classification, other data such as dwell time, incident detection and even origin destination information can be obtained. Video can also be used to provide surveillance information on a roadway.

New capabilities added to the system – intelligent video device and networking which offers new possibilities to the video-based traffic monitoring applications including:

- Getting access to complex intelligent system that will widely extend capabilities of particular photo-enforcement systems.
- Getting new emerging view of traffic flow and other parameters due to intelligent visualization of received data.
- General educational influence on drivers – for this purpose UnicamNet can serve in many ways. Besides red-light violence watch itself it can tell to drivers at a big display how quickly they have passed through some segment.
- Comprehensive information on the use of transportation facilities in urban areas provides the basis for many of the decisions made regarding the transportation infrastructure.

Both hardware and software parts of the system implementations were presented. Concept of the kernel design turned out well during the implementation of the application based project. There is also a possibility of expansion – even though new functions must be identically implemented on both platforms. Network communicating unit, which is a part of the kernel, allows to create any type of an application that requires communication. Module allows working with protocols of the network layer IP: UDP, TCP and a major part of the ICMP. ARP and the DHCP are considered to be also implemented. However, implementation of the communication layer is limited. It is not conceived for creating pure network devices such as switches, firewalls and routers.

Wide scalability of the system is achieved using a unified communication and networking subsystem called the Channels, which covers all the intra- and inter-board communication, as well as the communication between larger compact parts of the networked system and with the outside world. The usage of the standard TCP/IP stack and GPRS based mobile technology makes the communication technology cheap and reliable. It also allows remote debugging and monitoring of the system without a need for setting up any proprietary communication lines or forwarding monitoring data – the monitoring station can be an integral part of the system.

The advantage of the proposed system could be seen for example in the fact that it's based on cameras and machine vision algorithms only, comparing to others which are using additional sensors.

10. Acknowledgements

This work has been supported by the EU-HLT grant 506811-AMI – Augmented Multi-party Interaction.

11. References

[1] http://www.camea.cz

[2] Crha L., Fučík O., Zemčík P., Drábek V., Tupec P.: Inter chip communicating system with dynamically reconfigurable hardware support, in Proceedings of the 6th IEEE International Workshop on DDECS 2003, p. 311-312, Poznan, Poland, ISBN 83-7143-557-6.

Particle rendering engine in DSP and FPGA

Pavel Zemčík, Adam Herout, Luděk Crha, Otto Fučík, Pavel Tupec
Faculty of Information Technology, Brno University of Technology
Czech Republic
{ zemcik | herout | crha | fucik | tupec } @fit.vutbr.cz

Abstract

This paper presents an algorithm for rendering 3D point-clouds, which exploits an FPGA chip coupled with a DSP processor on an experimental board. Point-clouds are sets of graphical data in 3D space which seem to be more suitable for potentially many purposes than the most frequently used triangle meshes. The actual experimental implementation,, which verifies the concept and reports promising results, is also described.

1. Introduction

In recent years, the emergence of affordable 3D scanning devices along with the demand for even more geometric detail and rich organic shapes has created the need to efficiently process and render very large point sampled models. At data sizes where triangle based methods approach their limits, point representations are receiving growing attention [1][2].

While it is nice that the application developers can rely on the presence of accelerated graphics engines in the computers, it is quite unfortunate from the point of view of graphics and imaging algorithms research that the function of the graphics accelerators is usually quite strictly limited to rendering of planar triangles/polygons and limited choice of shading and texture algorithms and it is usually impossible to use them for implementation of any other algorithms. At the same time, the applied research of such high-performance graphics subsystems that leads in real products is being done by the manufacturers of the graphics subsystems and by the limited quantity of affiliated institutions, such as research laboratories and universities.

A reasonable way forward in graphics rendering architectures for those who do not directly collaborate with the graphics hardware manufacturers was offered by the recent development of Field Programmable Gate Arrays (FPGAs). Current technological progress allows implementation of even very complex devices in the programmable logic devices and achieve good results even with architectures and algorithms that are not supported by the traditional computer graphics manufacturers [2].

The hardware architecture for real-time high quality rendering of point-based graphical scenes is presented in this paper. By the point or particle we mean a surface element (also referred to as surfel) defined by x,y,z coordinates, nx,ny,nz normal, size and color. The design is based on an FPGA chip, hosted on a multi-purpose board featuring the FPGA chip, DSP processor, DRAM and SRAM memory. Common graphical accelerators (constructed to efficiently render polygon-based entities) are unsuitable for this purpose, since they don't offer any good way of transferring simple point/particle data. Transfer of triangle vertex data is effective enough (rasterization algorithms are far more time consuming than the transfer itself), but rendering points using this common hardware faces the bottleneck of data stream bandwidth [1].

1.1. Rendering algorithm

The rendering algorithm processes point cloud from virtually any data source, transforms the particles using a 3D transform matrix and evaluates the lighting intensity based on the positions of the light sources and surface geometry and material features. These processes can, however, in most cases be replaced by matrix multiplication and access into a table of pre-calculated values.

Finally, the particle shape is rasterized into the frame colour buffer according to their visibility determined by the z-buffer. The frame buffer is a 2D matrix, where each cell represents a pixel on the screen of appropriate bit-depth. The z-buffer is a similar matrix which contains for each pixel the z-depth (distance from the camera), so that for any particle being painted the system can determine, whether it is closer than the current contents of the colour buffer and should therefore be considered visible and painted or not.

Probably the most feasible geometrical representation of particles is circles. Their parameters are derived from the corresponding particles (see Figure 1).

Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04)
0-7695-2125-8/04 $ 20.00 © 2004 IEEE

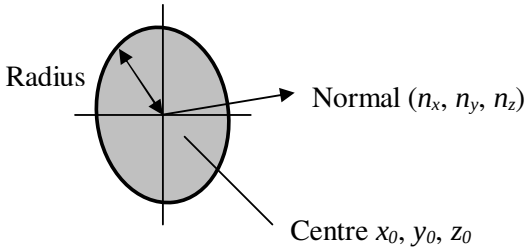


Figure 1: Particle projection

2. Rendering engine

The particle rendering engine being described is only a simple implementation of the proposed rendering architecture. It can be seen rather as a „proof of concept" than as the full-scale implementation. For this reason, maximum possible rendering subtasks tasks were left on the host DSP processor. This approach does not reduce the graphics subsystem throughput of the system but reduces the capability of the DSP of performing tasks, such as animation or simulation, related to rendering. The rendering subtasks currently implemented in the DSP that can eventually be moved into the FPGA include:

- 3D transformation (projection) of the particles,
- conversion of the particle normal vector (and lighting information) into the particle color,
- shape calculations (ellipsis parameters) based on the particle normal vector and projection.

The block diagram of the rendering engine is shown in figure 2. It is based on the above constraints and uses the DSP as the host processor that handles the particles and performs the above rendering subtasks. The DSP then transfers the particle data into the FPGA in the form of block memory transfer. The particle data comprises of the co-ordinates, encoded shape (described below), and colour information.

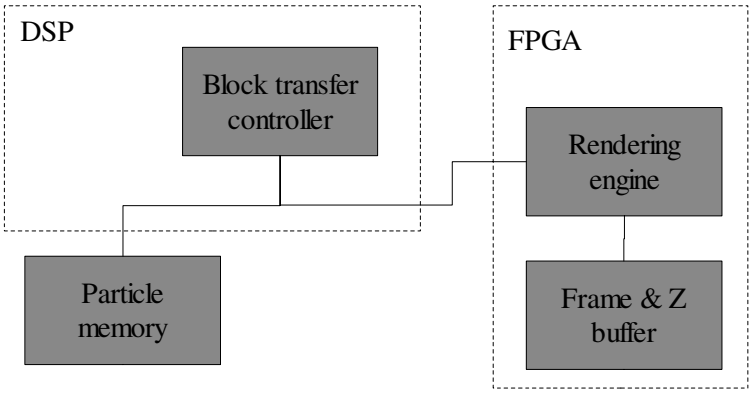


Figure 2: Rendering engine block diagram

3. Particle rendering memory requirements

As particles contain large numbers of pixels, it is desirable to have the frame and z buffers distributed in several memory banks that can be accessed in parallel. The distribution should be done in such manner that ensures that each rendered particles' pixels are stored in several of the memory blocks so that the speedup through parallel access in the memory is achieved. Simple yet efficient distribution of the memory is shown the figure 3. The raster image (output buffer) is divided into lines that are placed in the buffer banks individually. Let *N* be the number of the buffers. Every *N*-th line is placed in the same buffer bank while adjacent lines of the image are placed in different buffers whose number is defined by the line number *L mod N*.

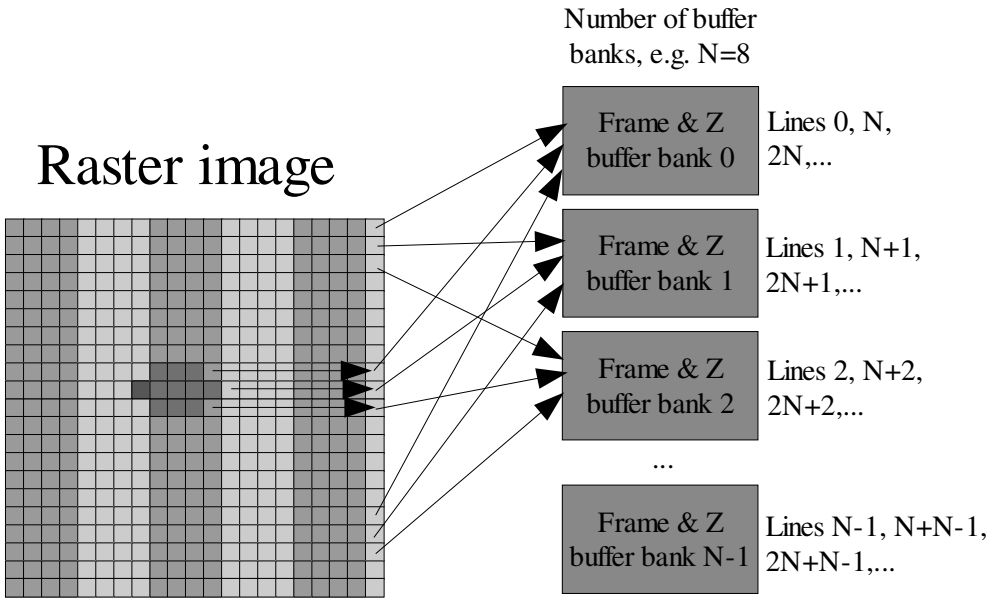


Figure 3: Rendering engine memory requirements

As shown on a simple example of particle (in red) in the image, the number of the buffers the particles' pixels belong to is the same as the number of the scan-lines of the particle S if $N \geq S$. Obviously, the *mod* operation is easily implementable if $N=2^n$ (where n is a suitable integer number). Also, if the raster buffers are organised in wide words, e.g. 32-bits wide and the pixels and depth ("z" co-ordinate) is encoded in e.g. 8 bits (this case is shown by grey stripes in Figure 3), several pixels (P pixels) of the raster can be placed in a single word of the raster buffer bank.

If the particle size is limited, it can easily be calculated how many memory banks B and how many words in each of the bank W are affected by the particle rendering. These figures also define the maximum requested memory cycles for particle rendering (storage) in the buffer banks.

In our case, we choose the maximum particle size 8x8 pixels. The maximum affected number of banks B=8 and maximum affected words in each bank is 3, so if the banks are accessed in parallel, the required memory cycles per particle is 3.

Unfortunately, the idea of distributed memory buffers (full-frame) is implementable only if:

a) the memory can be placed in the FPGA in full, or
b) the N memory blocks are implemented in hardware with separate access circuits.

However, none of the conditions are normally fulfilled or feasible to fulfil through design changes. In the theory, it would be possible to find FPGA with large enough memory buffers but it would be prohibitively expensive. It is also possible to create several memory access circuits for external memory access by FPGA, however, their number might not be enough (e.g. 8 not suitable) and this solution would also be quite expensive and as the external memories tend to be slower than internal ones, it would also degrade the performance.

For the above reasons, some suitable solution should be found that would preserve the advantages of distributed memory banks (for bandwidth) and also keep the solution simple and feasible. The solution to the memory access problem is "striping" - subdividing the projected particles into horizontal stripes. If it is ensured that the projected particle will fully fit into a stripe – horizontal subimage as shown in the Figure 4, the frame buffers requested should only convert the stripe and do not have to extend to the full image. If the rendering of the whole set of particles is converted into a sequence of rendering of the subset of particles that fit the particular stripe, and "flushing" the stripes into a simple (possibly external and low bandwidth) frame buffer, the performance should be preserved and the feasibility of the solution as well. The price is sorting of the particles according to the vertical (y) co-ordinate.

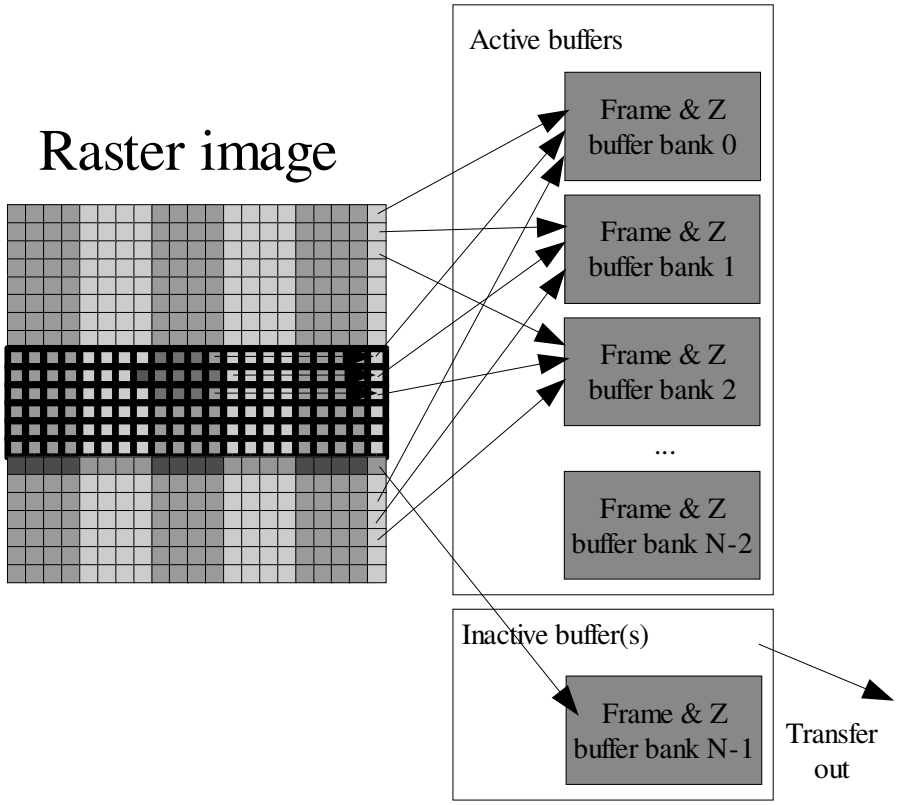


Figure 4: Rendering engine memory stripes

In the actual implementation, the rendering process is not interrupted by "flushing buffers" phase but the number of buffer banks is rather expanded so that the "active" banks – banks into which particles from current stripe may be drawn – actually form only a subset of all buffer banks and inactive banks' (at least one) content can be transferred into the external raster buffer along with particle drawing (one inactive stripe shown in green in figure 4). In our case, 16 banks total was selected for implementation while 8 are active and 8 inactive and may be constantly transferred into the frame buffer.

4. Shape encoding

The shape of the rendered particle is a projected circle – an ellipsis. The ellipsis is parameterized by the direction of the particle (normal vector of the circle plane) and by the size of the particle. While the shape of the ellipsis can be calculated through a relatively simple hardware circuit (see e.g. [4]), for small size particle projections it is also possible to tabulate the shapes of the ellipses. The task of tabulation of the ellipse shapes can be divided into two parts:

- decision what parameters will be used in the table,
- decision on how the shape will be represented in the table.

These parts can be dealt with relatively separately.

4.1 Rendering parameters

The shape of the particle is dependent on the normal vector *v* and size *s*. The normal vector represents only the direction of the particle and therefore the normal vector can be normalized (to have unit size). Therefore the vector also can be fully represented using only two of its components as for the normalized vector

$$v_x^2 + v_y^2 + v_z^2 = 1$$

so e.g. first two components of the vector represent it fully. For the small size of particles, it is suitable to just allow a small number of different values of these components, e.g. 64 in our case. The size of the particle is a scalar value and again, because of the small size of

the particle in pixels, it is suitable to allow only a small number of different sizes, e.g. 16 in our case. So the total number of items in the table is in our case 64*64*16=65536 items.

4.2 The shape representation

The ellipsis that is a projection of the particle is symmetrical. It is assumed (as stated above) that the particle projection fits into an 8*8 pixels square. Because ellipsis is a convex object, it is clear, that if the shape is encoded line-by-line, only one continuous stripe of pixels in a line - "scan" is needed for each line, see figure 5. As the projection fits in 8*8 pixels and is symmetrical, all that is needed is to encode 4 lines. The code for each line can consist of "scan start" and "scan stop" (see figure 5).

Start and scan information can easily be encoded into 6 bits. 3 bits are used for direct encoding of the start code while other 3 bits for stop code. If stop code is smaller than start code, the code is invalid and represents empty scan. The symmetry operation is easy to implement by swapping and bitwise inverting the start and stop codes.

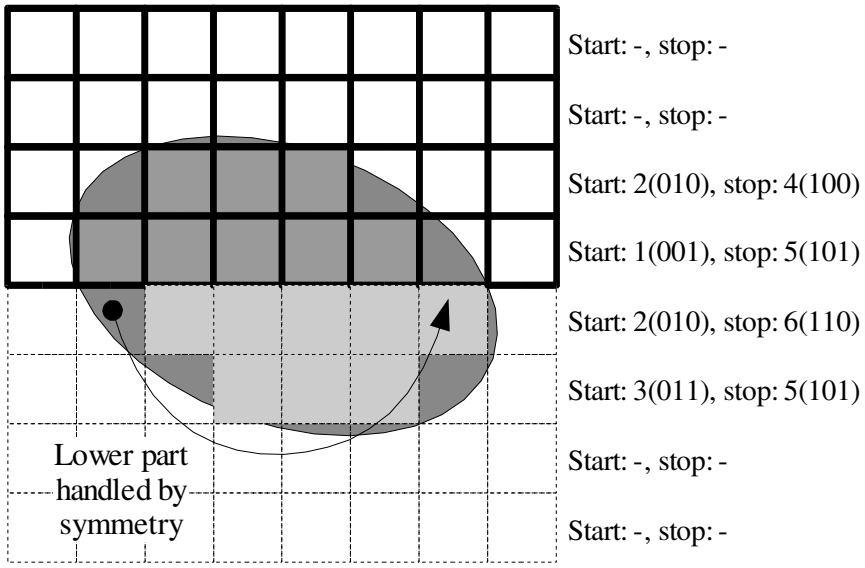


Figure 5: Ellipse shape encoding

5. Experimental hardware implementation

For the hardware implementation, FPGA Xilinx Virtex E 300 is used. In the future, Virtex II is planned to be used instead. Hardware design programmed in the FPGA consists of particle reader/writer, pixel reader/writer, frame and Z buffers and the viewing engine. FPGA input frequency is 100 MHz, but for the major part of the design, 50 MHz is used. Accessing time to the SRAM (used as a video-RAM) is 15 ns. This memory and ADV 478 chip (D/A converter and palette memory) are placed outside the FPGA. There is also 16 MB SDRAM placed at the board used by the DSP. This SDRAM runs at 100 MHz.

5.1 Display subsystem

This subsystem takes care about correct viewing of an image placed in the SRAM and its writing into this memory.

Every pixel clock period, data are read from the memory (address is the counter automatically incremented every pixel clock cycle). Meanwhile, shared data bus is put to the third state at the side of FPGA, so that data from the memory could be read by the ADV 478 chip. Pclk rising edge ensures the data on this bus to be converted to analog format suitable for a TV or a monitor.

Between every two pixels read for the TV out, it is possible to place one read or write cycle from/to SRAM. This is used for reading the image to the DSP and writing pixels into the RAM. This writing comes from

two sources – pixel writer (pixels written directly from the DSP) and frame buffer of the particle writer (pixels written by the rendering engine).

SRAM is organized as a two-bank video RAM in order to implement double-buffering: while writing an image to one bank, the second bank is being shown on the screen and vice versa.

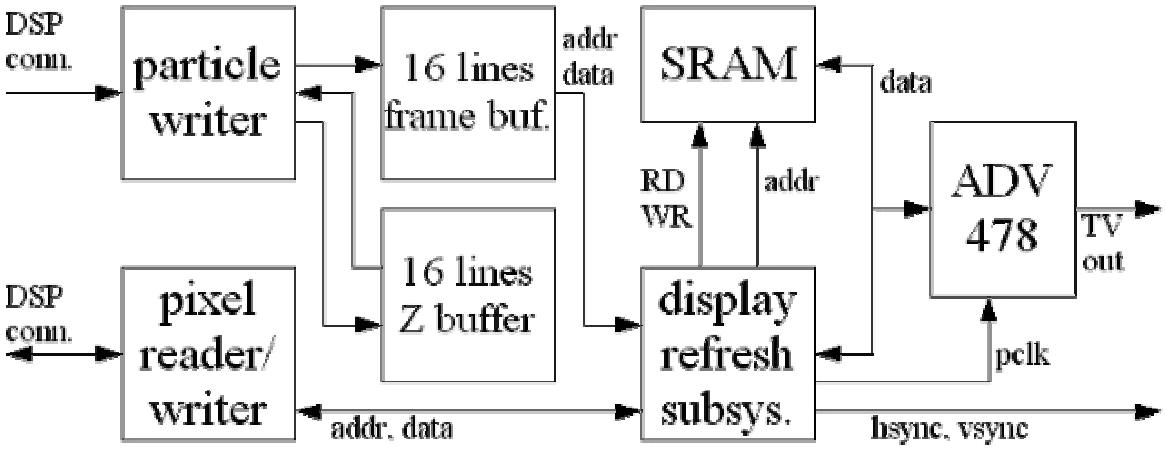


Figure 6: Rendering and display subsystems

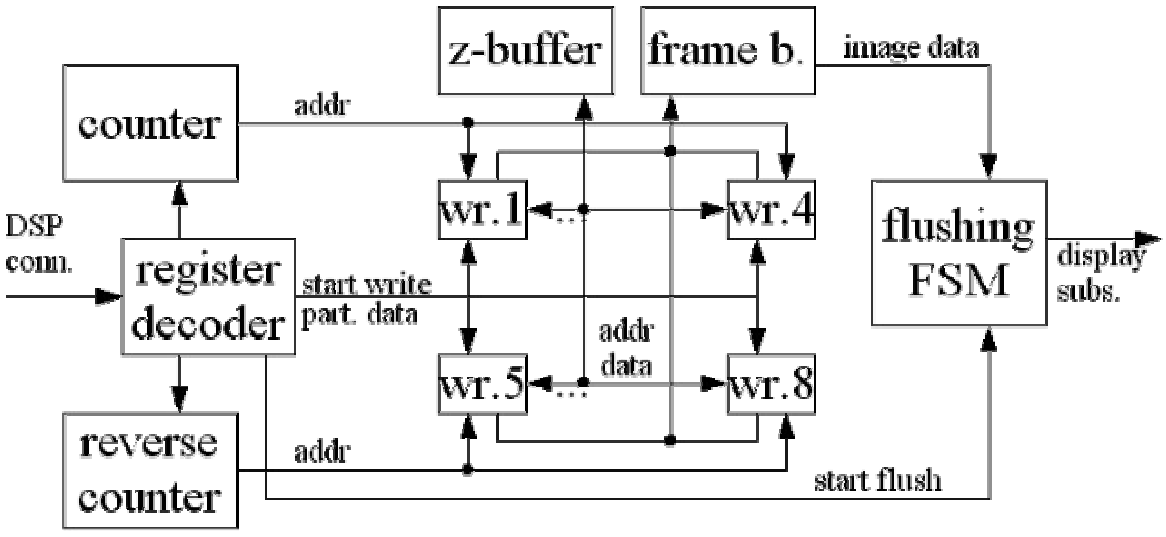


Figure 7: Rendering engine detail

5.2 Pixel reader/writer

This unit simply gets data and address from the DSP and makes a write to the SRAM through the viewing engine. This may be used for writing some additional information to the screen. It is also possible to read the data from the memory. It allows to read the image to the DSP and store it for future use.

5.3 Particle writer

Functional description of this unit is described at the theoretical part of this article. Hardware implementation consists of few state machines using two groups of block RAMs – one for the frame buffer and one for the z-buffer.

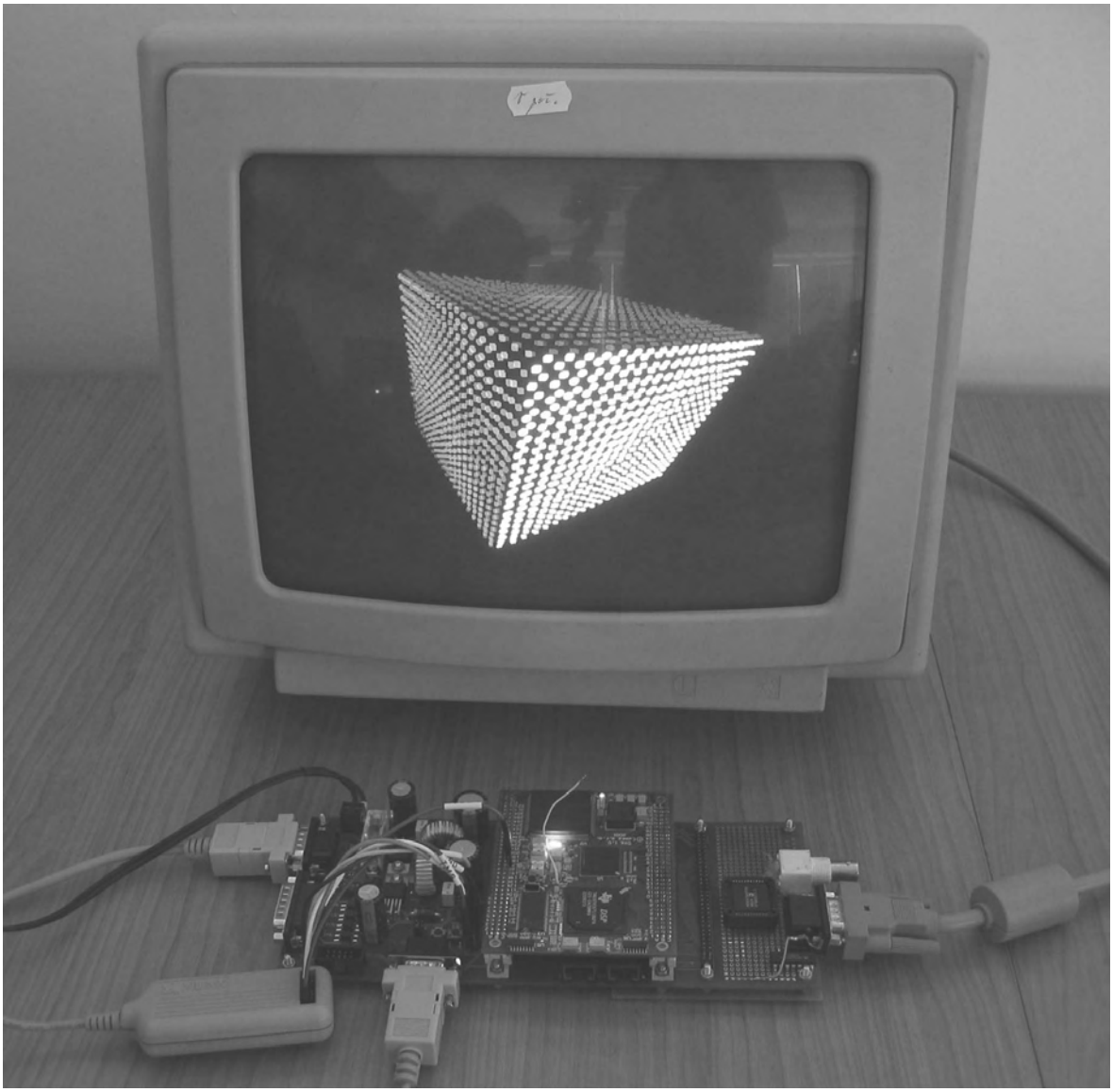


Figure 8: Experimental setting running with simple data set

Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04)
0-7695-2125-8/04 $ 20.00 © 2004 IEEE

When a particle description comes from the DSP, it is processed by the register decoder. Base horizontal coordination is set to its position, and two counters are running to define the exact position of the current processed pixel. One counter runs from zero to maximum and displays the upper part of the particle. Second counter runs from maximum to zero and displays the lower mirrored part. Writing engines start writing reacting to the start write signal. While processing, the data is read from the z-buffer and the writing engines decide whether to write (both to the frame and the z buffer) or not by comparing the actual depth with the depth from the z-buffer. Reading from the z-buffer must start some cycles before the writing process due to the memory read latency.

When the special code word is written from the DSP, writing is moved to the next line. Flushing of the processed line is started – data from the block ram are written to the SRAM through the viewing engine.

6. Achieved results

Current maximal number of particles shown by the FPGA is 5 million per second. This number comes out from the clock period which is 20 ns, and the number of cycles required for showing one particle. One column of a particle is written in one period, and two periods are required for the pre-reading the z-buffer data. Totally 10 periods are 200 ns per particle.

Of course there are many possibilities how to improve the performance. Using more powerful FPGA chip (for example Virtex II) would lead into higher possible frequency (we assume at least 100 MHz). Other possibility is to parallelize writing to the memory by setting the width of the data bus to the block-rams from 8 bits to 32 bits. Extra logic for treating this situation would be needed, but speed-up ratio would be up to four. We could also avoid the z-buffer reading latency by pipelining. Finally, we could show one particle in 2 clock cycles (10 ns), which means speedup up to 10 times from the current state to 50 million particles per second, still using standard off-the-shelf components

7. Conclusions

In this paper an algorithm for rendering of huge point clouds has been presented. Point-based computer graphics seems to be a concept of close future for visualization and realistic rendering, partially replacing the most common approach at the moment – triangle meshes.

The proposed rasterization algorithm solves the rendering task including the visibility issues between the particles. It is designed to be implemented using off-the-shelf programmable logic components of low cost. Major part of the projection phase is left to the superior controlling task being performed by a signal processor. This leaves space for further hardware acceleration.

Hardware implementation in FPGA consists of the subsystem treating read and write cycles of the video-RAM, pixel writer and the particle writer. This unit is the heart of the system – it consists of eight pixel writers that write the data to the internal block rams and a flushing state machine for transferring an image to the video RAM.

Current speed of drawing the particles is 5 million per second. Changes are proposed that could increase this number up to 50 million still using standard off-the-shelf components. However, this implementation is considered to be rather a proof-of-concept than a final graphics acceleration solution. The Virtex II Pro Xilinx FPGA that is to come should allow further optimizations and may be ground for a graphical hardware challenging graphics equipment of desktop computers.

8. Acknowledgements

The research presented in this paper has been supported by the EU-HLT grant IST-2001-34485 “Multi Modal Meeting Manager”.

References

[1] Gross, M: “*Point Based Computer Graphics”, Spring Conference of Computer Graphics 2002*, Budmerice, Slovakia, 2002

[2] Rusinkiewicz, S: “*Surface splatting”, Proceedings of SIGGRAPH 2001*, USA, 2001

[3] Zemčík, P: “*Hardware Acceleration of Graphics and Imaging Algorithms Using FPGAs”, SCCG 2002*, Budmerice, Slovakia, 2002

[4] Zemčík, P, Tišnovský, P, Herout, A: “*Particle Rendering Pipeline”, SCCG2003*, Budmerice, Slovakia, 2003

[5] *VirtexTM 2.5V Field Programmable Gate Arrays, Xilinx, DS003-1 (v2.5)*, April 2, 2001, USA, 2001, (available at http://www.xilinx.com)

[6] *TMS3B0C6711, TMS320C6711B Floating point Digital Signal Processors, Texas Instruments, SPRS088B,* September 2001, USA, 2001, (available at http://www.ti.com)

SCIA '97 The 10th Scandinavian Conference on Image Analysis
Lappeenranta, Finland, June 9-11, 1997

Visual Analysis on the Production Line

P. Zemčík
Department of Computer Science and Engineering
Faculty of Electrical Engineering, Technical University of Brno
Božetěchova 2, CZ-612 66 Brno, Czech Republic

P. Valenta, J. Honec, M. Richter
Department of Automation and Measurement
Faculty of Electrical Engineering, Technical University of Brno
Božetěchova 2, CZ-612 66 Brno, Czech Republic

O. Fučík
Department of Computer Science and Engineering
Faculty of Electrical Engineering, Technical University of Brno
Božetěchova 2, CZ-612 66 Brno, Czech Republic

Abstract

This contribution describes the system for visual analysis of small electronic components on a production line. The system has been successfully implemented and is currently in use. The focus of this contribution is on imaging algorithms; however, a brief overall description of the system is also presented.

1 Introduction

The demand for industrial applications of image processing and computer vision is growing quite rapidly and so is the ability to implement the applications. This contribution illustrates one example of successful realization of the industrial application of visual analysis [1].

This contribution describes visual system that has been implemented in co-operation with CAMEA Brno, Czech Republic, a company that specializes in computer imaging, and EMO Brno, Czech Republic, an optical company, for KOMFI Lanškroun, Czech Republic, a company that specializes in packaging machines for small electronic components.

The task was to implement real-time system for visual checking of the quality of components as they are inserted in the plastic carrier belt used at the customer's side directly by SMD mounting machines.

1.1 System Requirements

The packaging machine usually cuts the components from a lead frame (each containing ~50 of components), bends their leads, checks electrical functionality, and checks, whether the components belongs to one of the pre-defined classes depending on their electrical tolerances. Based on the above information, the components are then inserted in one of the two carrier belts (that contain thousands of components), which are then sealed. This process is repeated in approximately 500ms cycles.

During the cycle, visual system must check the bending of the leads and the print on the component body. Unfortunately, due to mechanical construction of the packaging machine, the visual system is forced to check prints on up to two components in the same cycle; therefore, the visual system should be capable of checking one bending and two prints in one cycle.

2. System Description

The visual system consists of three image acquisition units and Digitizing and processing unit. The image of the components goes through the lenses to video cameras. The data from the video cameras is digitized in the Digitizing and processing unit and is also processed there. Integration with the packaging machine is accomplished by connecting the Digitizing and

processing unit of the packaging machine controller, which decides what to do if the component is out of the tolerance limits. See figure 1.

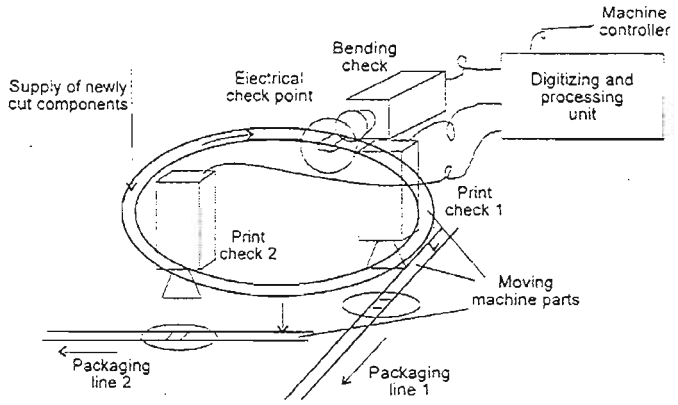


Figure 1: Visual system sketch

2.1 Image acquisition

The image acquisition is performed by the specialized optical system (designed to order by EMO Brno, a specialized optics company) and a proprietary video grabber. The optical system consists of two types of lenses with built-in LED illumination systems. One type of lenses is assigned for lead bending (high contrast side view) and the other for print capturing (top view). Both lenses were optically quite complicated due to their optical gain close to 1, which is the most complicated case to handle. Moreover, the physical limits imposed by the existing packaging machine forced the lead bending optical system to be bend by 90 degrees and use a mirror in the middle of the optical system.

The video grabber is proprietary designed for low noise and high geometrical precision that is achieved by pixel-synchronized grabbing. The maximum resolution of the images is approximately 740x285 8bit pixels while in most parts of the system only half horizontal resolution is used (approximately 370x285), which leads to image sizes around 100KB. The video grabber is connected to the host computer via the ISA bus interface.

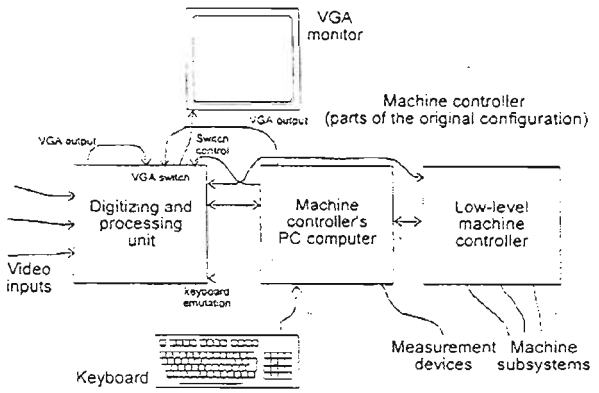


Figure 2: Integration of digitizing and processing unit in the system

2.3 Communication

Communication between the digitizing and processing unit and the machine controller is performed solely using 24 Volt logic signals. Table 1 describes the used signals (signal types are marked from the view point of the digitizing and processing unit).

Signal name/type	Signal description
DI/input	General data communication input
DO/output	General data communication output
CI/input	Clock input for data communication
CO/output	Clock output for data communication
MN/input	Processing unit manual control and VGA switch signal
OK/output	General status output
SB/input	Start bend grabbing and processing command input
BO/output	Bend output result of processing
SP1/input	Start print 1 grabbing and processing command input
PO1/output	Print output 1 result of processing
SP2/input	Start print 2 grabbing and processing command input
PO2/output	Print output 2 result of processing

Table 1: Description of communication signals

Most of the above input signals are generated by the machine low-level controller, which also accepts the output. Exceptions are the data communication signals that mutually connect the PC computers in the system. An example of timing of the signals can be seen in the figure 3.

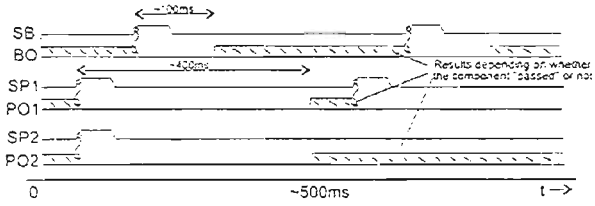


Figure 3: Input/output timing diagram (see table 1 for signal description)

3. Imaging Algorithms and Methods

The main target during design phase of the imaging algorithms and methods was to make them as reliable and as fast as possible. The reliability requirement was enforced by the fact that the system had to be designed to check the quality of components that has to be ensured by their producer so that depending on the contract only units of components per million could be out of specifications.

The imaging methods used in this application were analyzed and later empirically checked and fine-tuned for the best possible time-efficiency and reliability. In some cases, more sophisticated methods, that were also considered, had to be given up for insufficient computational power of the host PC computer.

The imaging algorithms consist of the algorithm that checks the lead bending and a set of algorithms that check the component's logo and label.

3.1 Lead Bending Check Algorithm

The lead bending check algorithm checks the shape of the component's leads using the side view of the component just before the electrical measurement takes place. The required resolution of the measurement is ~1/100mm while the length of the component is several millimeters Taken into account the horizontal resolution (370 or 740 pixels), it was necessary to use sub-pixel resolution methods. Therefore, the optics and illumination were adjusted so that the contrast of the edges of the component and leads is high, which enlarges the useful range of intensity. See figure 4.

The purpose of the lead bending check is to measure relative positions of pre-defined points on the leads. The practical purpose and reason why the measurement is important is that during the SMD mounting process the component is glued to the PCB first using the gluing blob shown in the figure 4, so the gluing blob must be accessible and close to the PCB surface. Then the component is soldered and this process requires that the leads are very close to the PCB surface, too, and that the size of the component is in a very narrow interval.

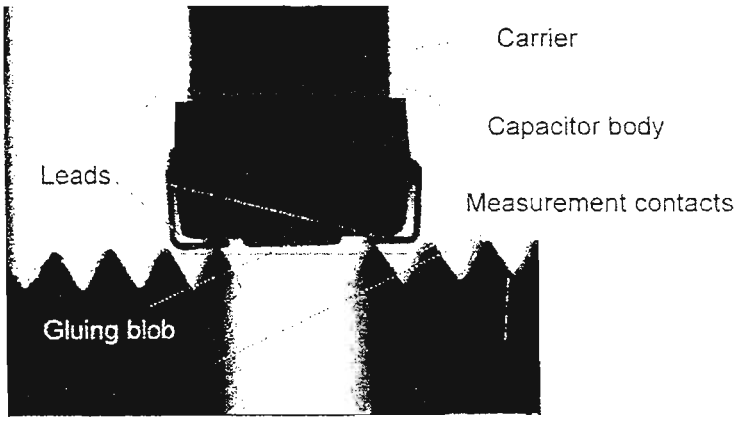
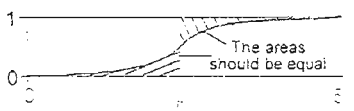


Figure 4: Side view of the component

3.1.1 Subpixel edge detection

The placement of the component in the image is not varying very much so that no specific component body searching algorithm is necessary. Also the approximate direction of the edges is known. The position of the edges is further localized in more detail using the highest gradient principle. After the position of the edge is found with precision of ~1 pixel, the subpixel algorithm is started on a slice, approximately orthogonal to the edge, based on the equation (1) solved numerically.

(1) $\int_0^p I(x)dx = \int_p^s 1 - I(x)dx$, where x is the position, and p unknown



The above equation (1) is solved for several points along pre-defined parts of the lead that are assumed to be straight. Small pieces of dust can, however, spoil the edge. To avoid erroneous result in this case, linear regression is performed on the calculated points and those that are in the highest distance from the edge are rejected. Then the linear regression in repeated and if successful, the result is considered accurate. An example of images of accepted and rejected components are shown in the figure 5. The algorithm consumes approximately 15ms of the CPU time that is acceptable from the view point of the overall system timing.

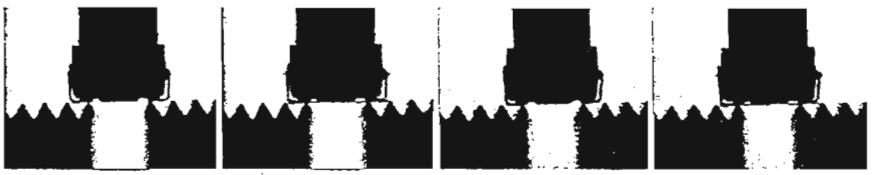


Figure 5: Accepted (left) and rejected (right) components prior and after processing
(component on the right is rejected due to the gluing blob having no contact with the PCB)

3.2 Print Check Algorithm

The print check algorithm detects the orientation, texts, and logo of the component just before it is sealed into the plastic carrier belt. The steps involved in this process are detection of the position and orientation, "normalization" of the component's position, checking the logo, thresholding and checking the texts, and checking the integrity of the component's body and amount of dust on the component body. The top view of the component used for print check is shown in the figure 6. Note that two identical print check subsystems exist in the system. It should also be noted, that the shape of the characters on the component (but not its contents) can vary, which forces the checking algorithm or to check the contents on a high level (character recognition) or to be able to check the shape of the texts to several templates during one check.

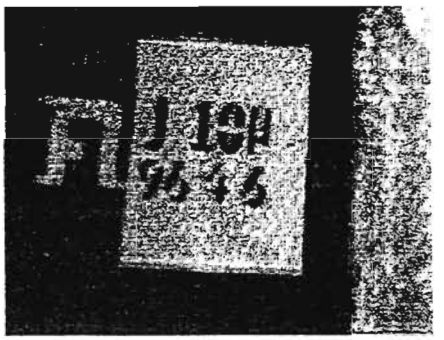


Figure 6: Top view of the component

The purpose of the print check can be subdivided into the following items:

- Orientation - the components are polarized and no further check is done before soldering
- Presence and quality of the logo - this check also detects placement of the texts
- Texts - the measured value should correspond and also the time stamp must be accurate
- Integrity and dust - the component's body must not be broken and must be clean

3.2.1 Position and orientation

First step in print checking is finding the position and orientation. For the purpose of this step the component could be seen as a rectangular object of the known size. Unfortunately, the component's edges image is distorted due to irregularly reflected light. This was also the reason why several simple algorithms (such as edge detection in thresholded image, interpolation using the least square criterion) failed when applied to this task.

Suitable method proved to be the Hough transform. The Hough transform in this case transforms the image space *(x,y)* into the straight line space *((k,q),(K,Q))*, where

(2) $(x, y) = (k \cdot x + q, K \cdot y + Q)$, where *(x,y)* is the image space

The inherent Hough transform algorithm whose inherent high complexity usually leads to long execution times was simplified using the following methods:

- Subsampling the image (using only a few slices of the image in each direction) - this was possible only due to huge dimensions of the component in the image and due to the fact that the component's shape is known and precise.
- Evaluating the equations only for a small range of angles - it is known that the component cannot be rotated by more than approximately ±20° due to the physical constraints of the carrier belt.

It was empirically tested that simple search for the maximum value of the result of the Hough transform is sufficient to localize the component precisely enough for further processing, so that more advanced methods [2] need not be used. Image showing the result of the Hough transform for the horizontal edges in the image can be seen in the figure 7.

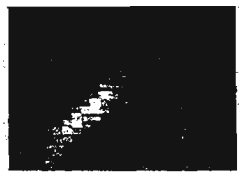


Figure 7: Hough transform result (horizontal edge pre-processed using edge detection)

For the next processing, the component's image is rotated so that its edges become parallel with the *x* and *y* axes in the image. The whole process consumes approximately 30ms of CPU time.

3.2.2 Logo and Text Detection

The logo and text are detected using intensity slices and correlation techniques. The logo is relatively easy to detect as its background is dark as opposed to light background for text. Moreover, The logo's direction is very close to the image axes which makes the process easy. The final solution was to horizontally slice the image, and search for the edge in the mean of the slices.

Different situation, however, is with the texts. It would be possible to perform 2D correlation with a reference text but this would also be time consuming. The solution was to average several slices of the text in vertical direction, find the vertical position of the texts and then localize the text horizontally using correlation of a few slices of the text with a reference text. The view of a component with the texts localized can be seen in the figure 8.

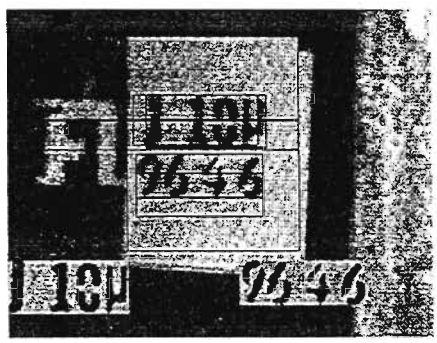


Figure 8: Rotated component with localized and thresholded texts

3.2.3 Text Check

Several methods were tried to check the printed text. This included correlation with character templates, checking for the text features using graph grammars, and a few character recognition methods used in OCR. None of the methods above matched both the requirements for speed and reliability at the same time; therefore, another, relatively simple method, was tried.

The method that has finally been used is to threshold the image first to get a binary image of the texts, then XOR the text image with a pre-recorded template, and finally to erode the result using a morphological filter and count the number of "ones" to decide, whether the text is "close enough".

The thresholding algorithm with fixed threshold failed. The reason for this was that large spots with varied intensity exist on the components. Therefore, local thresholding had to be used. First attempt was to consider the threshold the mean of minimum and maximum in a small neighborhood (while omitting random erroneous values). This approach worked well, but proved to be too slow. Therefore, an alternative was searched that calculates the local threshold T using some non-linear function of pixel values. The following function was found satisfactory:

(3) $T = I_A + \frac{1}{2}\sqrt[3]{\frac{1}{n}\sum_{x=1}^{n}(I_x - I_A)^3}$, where $I_A = \frac{1}{n}\sum_{x=1}^{n} I_x$ (note that $\sqrt[3]{\ }$ preserves the polarity)

The above expression was further simplified so that it could operate on sums of the pixel values, and their second and third power. This made possible to use many intermediate values from the consequent (overlapping) areas in the image repeatedly. See equation (4).

(4) $T = I_A + \frac{1}{2}\sqrt[3]{\frac{1}{n}\sum_{x=1}^{n} I_x^3 - 3I_A\frac{1}{n}\sum_{x=1}^{n} I_x^2 + 3I_A^2\frac{1}{n}\sum_{x=1}^{n} I_x - I_A^3} = I_A + \frac{1}{2}\sqrt[3]{\frac{1}{n}\sum_{x=1}^{n} I_x^3 - 3I_A\frac{1}{n}\sum_{x=1}^{n} I_x^2 + 2I_A^3}$

To speed the calculations up further, all the calculations are performed on integers and the square root is approximated by evaluating first three steps of the Newton iteration.

As mentioned in 3.2., it was necessary to compare the image with several templates. Up to 8 templates could be compared to the image by storing the thresholded image back in the byte array so that logical 1 correspond to "all ones" and logical 0 to "all zeros". The templates are stored so that each template is stored in one "bit plane" of another byte array. The XOR is then performed on the byte array and "erosion" operation, converted into series of logical operations, done on the same array. This leads to processing of several templates nearly for the price of one. See figure 9.

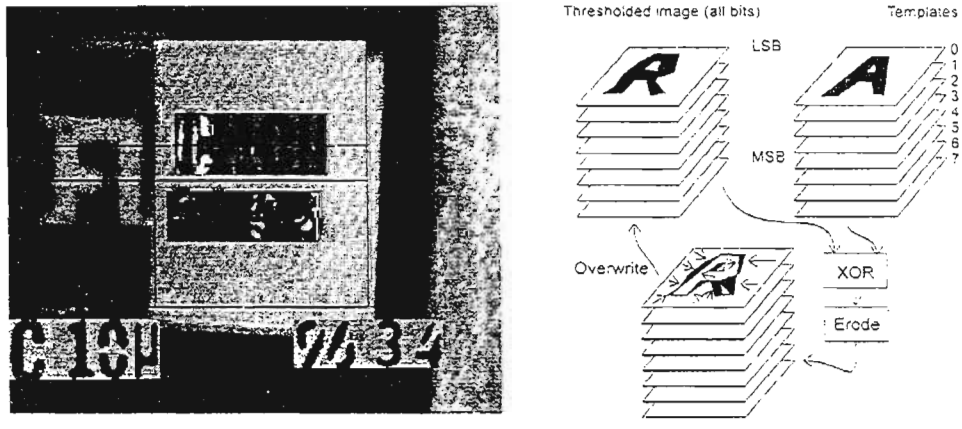


Figure 9: Text image (rejected) after XOR and erosion with the templates
(note that different combinations of 0 and 1 are shown as different gray levels)

During the erosion phase, the resulting erroneous pixels are counted and based on the lowest error count it is decided whether or not to accept the text image. The print check algorithm is the most time critical part of the whole process and consumes more than 100ms of CPU time.

4. Software Platform

The visual system software was written in Borland C++, version 3.1 under MS-DOS 6.22. This solution might seem obsolete [3] but at the time the development begun it was the only reliable solution on PC platform that ensured appropriate real-time control over the PC hardware features.

Several system extensions to MS-DOS were implemented, such as timer speedup, improved EMS handling, keyboard emulation, multithreaded core, etc.

5. Conclusion

This contribution shows an example of a real-life visual system for checking the quality of components dedicated as SMD; this analysis is done directly on the production line. Prototype of the system is fully operational, and it is currently ready for replication that will happen in next few months.

The system is now exploiting nearly all the available computational power of the Intel Pentium CPU running at 166 MHz so the possibility for using more complex algorithms is limited. It can, however, be expected, that if such improvements are be needed, the CPUs will be changed to better performing ones.

The visual system shows very high reliability in detecting failures in the components. The system has been in experimental use for several months but as the number of faulty components that were not detected by the system is nearly yero, it was so far not possible to produce proper statistical evaluation.

Acknowledgements

The authors would like to thank KOMFI Lanškroun, EMO Brno, and CAMEA Brno, all from Czech Republic, for their significant help during the development of the system, Grant Agency of the Czech Republic, for financial support through GAČR 102/97/1012 grant, and Information Processing Group, Lappeenranta University of Technology, Finland on whose ground and with whose support this contribution was summarized and submitted.

References

[1] Honec, J. et al: Industrial Applications of Computer Vision, (Průmyslové aplikace počítačového vidění, in Czech, proceedings, Conference of the departments of automation, Brno, Czech Republic, Sep. 1996

[2] Kalviainen, H., Hirvonen. P., Oja, E.: Houghtool - A software package for the use of the Hough transform, Pattern Recognition Letters 17 (1996), Elsevier Science, The Netherlands, pp. 889-897

[3] Zemčík, P.: Image Processing Software in MS Windows, proceedings, 10th Conference Process Control. Tatranské Matliare, Jun. 95, Tatranské Matliare, Slovak Republic, Vol. 1, pp. 228-231

Project of IPv6 Router with FPGA Hardware Accelerator

Jiří Novotný[1], Otto Fučík[2], and David Antoš[3]

[1] Institute of Computer Science,
Masaryk University Brno,
Botanická 68a, Brno 602 00, Czech Republic
novotny@ics.muni.cz
[2] Faculty of Information Technology,
Brno University of Technology,
Božetěchova 2, Brno 612 66, Czech Republic
fucik@fit.vutbr.cz
[3] Faculty of Informatics,
Masaryk University Brno,
Botanická 68a, Brno 602 00, Czech Republic
xantos@fi.muni.cz

Abstract. This paper deals with a hardware accelerator as a part of the *Liberouter* project which is focused on design and implementation of a PC based IPv6 router. Major part of the Liberouter project is the development of a hardware accelerator – the PCI board called COMBO6 and its FPGA design which allows processing most of the network traffic in hardware.
Keywords. IPv6, router, Liberouter, Virtex II, FPGA.

1 Introduction

The paper describes a hardware accelerator based on FPGA that is designed to speed-up packet processing in a PC-based IPv6/IPv4 router. It is a part of the *Liberouter* project[1], which aims at developing a high-performance router with entirely open design.

Both the reliability and functionality of PC routers has proven to be fully comparable with commercial middle-class products [4]. Two main limitations keep from wider use of PC based routers. First, the PC routers have more difficult configuration than commercial routers. Second, the PC based routers have reached their theoretical throughput due to system resources saturation. Even the fast PCI bus (64 bit, 66 MHz) is not powerful enough.

The goal of the Liberouter project [5] is to develop an IPv6 PC based router, which solves both limitations discussed above. The configuration issues will be worked out by means of a uniform configuration environment based on XML. Performance will be improved using of a hardware accelerator that processes most of the network traffic in hardware.

[1] This research is supported by the FP5 project "6NET" (IST-2001-32603) and CESNET project "IPv6 implementation in the CESNET2 network" (02/2003).

P.Y.K. Cheung et al. (Eds.): FPL 2003, LNCS 2778, pp. 964–967, 2003.

2 Router Architecture

The hardware accelerator is a PCI card containing network interfaces, FPGA, memories (SSRAMs and DRAM), and necessary logic. The card has several expansion connectors dedicated for interface cards and future extensions [6]. All physical interfaces are mounted on an expansion daughter board which allows to use many different interface standards, either metallic or optical.

Packet switching and filtering itself will be performed by the accelerator. Software can do the rest, providing operations like routing paths calculations, router configuration, and statistics computing. Such operations are not time critical; and PC's resources are suitable for them. Communication through the PCI bus will be limited to board configuration, routing tables and firewall rules initializing and changing, statistics collecting as well as exceptions handling.

Moreover, if the accelerator behaves as usual network adapters from the point of view of the Unix system, we may use ordinary system mechanisms for routing and packet filtering tables maintenance. This way we obtain configurability of a software router and speed of a hardware one. The router configuration is available by means of usual software like `ifconfig` and routing daemons. Development of the PC based router with an accelerator requires the design methodology known as *hardware/software codesign*. We have begun with the PC based router fully implemented in software, moving more and more operations to hardware. We start from input and output stages and continue with more complicated blocks.

3 Packet Processing in Hardware

Packet processing in FPGA is done by a chain of dedicated processors – we call them *nanoprocessors* due the simplicity of their instruction sets. Each nanoprocessor has its own specialized instruction set designed for the particular purpose. The programs of nanoprocessors are stored in both FPGA's on-chip memories as well as in the external SSRAM memories.

Complexity of nanoprocessors lies "between a Finite State Machine (FSM) and RISC processors." The advantage of the this approach is the possibility to change functionality at runtime, as opposed to the case of FSMs. There is no need to rewrite the source code (e.g., in VHDL), synthesize it, place and route the design, and download the configuration data into FPGA. This differs from partial reconfiguration where all development steps must be done. On the opposite, partial reconfiguration can lead to smaller and more efficient design.

Let us now briefly describe the packet lifecycle in the proposed router. Complete information can be found in [1]. The packet processing is pipelined, the packet flows through the FPGA and memories. An incoming packet is received by the Input Packet Buffer and passed to the Header Field Extractor. The HFE pushes the body of the packet (including original headers) into the dynamic memory. Meanwhile, it parses the packet's headers and creates a *Unified-header* and a structure reflecting the actual arrangement of headers in the packet. The Unified-header is a fixed structure containing relevant information from packet

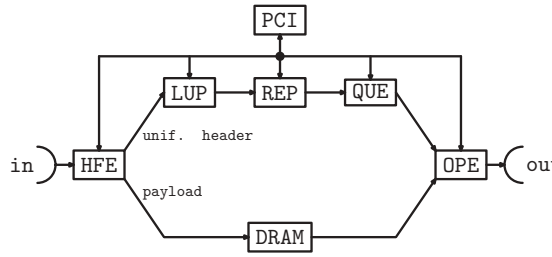


Fig. 1. Lifecycle of a packet traversing the router

headers, allowing to abstract the subsequent lookup operations from the actual header structure.

The Lookup Processor (LUP) uses CAM and SSRAM. LUP processes the Unified-headers by performing the lookup program and puts the result to Output Packet Editor. Using CAM is the fastest possibility when the application is small so that the lookup table fits into the memory. Unfortunately this is not the IPv6 case where we must match more that 440 bits of packet headers to decide what to do with a packet when it provides firewalling services.

Due to reasons discussed above we have developed a novel approach for the lookup machine. The word to be matched is stored in a sequence of registers. Instructions interpreted by the lookup machine are just conditional jumps. The lookup method is based on a (level compressed) search tree structure [1].

The Packet Replicator and Block of Output Queues (RQU) replicates the packet identification as well as the pointers to the editing programs into the dedicated queues. RQU computes the number of replicas of the packet that should be sent out. The Output Packet Editor (OPE) block modifies headers of packets. The OPE can also send a packet to the operating system, in the case when the packet destination is either the host computer itself, or the packet cannot be processed by hardware. This concept allows adding new features, step by step, during the PC based router development and use. The price paid for such approach include slower processing of software processed packets.

4 Software Support

Software drivers are developed first for NetBSD (FreeBSD) and ported to Linux. Driver operations include the FPGA chip configuration, accessing all memories mounted on the card as well as inside FPGA, and other hardware/software interface operations. The other part of router software should also be able to hide the card presence in the PC – the card should perform the same routing and filtering functionality as the operating system itself. The task is to develop a daemon which monitors the changes in both routing and filtering tables and, upon their update, it generates a nanoprogram for the LUP processor. To be able to make both routing and filtering by one searching operation, we have developed

a concept of *routing/firewalling table*. The routing/firewalling table contains filtering rules applied apriori to the routing table rows, it can be represented as a tree structure and converted to a LUP nanoprogram [1].

PC based routers are running under various operating systems with various configuration files. It causes problems to network administrators. To overcome this we are working on a unified configuration environment [3]. We have selected the XML language to store the router configuration. The user interface will be implementing Command Line Interface, web interface, and SNMP. From XML, the router configuration will be generated depending on the current operating system. This approach simplifies the use of the router and the user interface can be compatible with these seen in industry standard routers.

5 Conclusion

The PC based router design is a complex and long time task. It requires cooperation of experts from many areas. Currently, the entire team has more than 35 people from organizations in the Czech Republic including Masaryk University, Brno Technology University, Czech Technical University in Prague, and Camea ltd., as well as several consultants from other countries. The whole Liberouter project is organised by CESNET.

Currently, the accelerator prototype has already been developed, and produced. Processing blocks described are simulated and tested in hardware. The IPv6 PC based router is primarily dedicated but not limited for the use in academic networks, with focus on research in the internet protocols.

The project is fully open and all materials including source codes are available in the Internet [5]. Due to its flexible architecture, the hardware accelerator with low level software utilities can serve as a general purpose hardware platform for computation acceleration as well. There are many research project teams interested in using the proposed system. Possible application include programmable OEO (optic-electric-optic) switch, network monitoring tools, evolvable hardware research, reconfigurable computing, digital signal processing, and encryption.

References

1. David Antoš, Jan Kořenek, Kateřina Minaříková, and Vojtěch Řehák. Packet header matching in Combo6 IPv6 router. Technical Report 1/2003, CESNET, 2003.
2. Jiří Barnat, Tomáš Brázdil, Pavel Krčál, Vojtěch Řehák, and David Šafránek. Model Checking in IPv6 Hardware Router Design. Technical Report 8/2002, CESNET, 2002.
3. Petr Holub. XML Router Configuration Specifications and Architecture Document. Technical Report 7/2002, CESNET, 2002.
4. Ladislav Lhotka. Software tools for router performance testing. Technical Report 10/2001, CESNET, October 2001.
5. Liberouter. Liberouter Project WWW Page. `http://www.liberouter.org`, 2003.
6. Jiří Novotný, Otto Fučík, and Radomír Kokotek. Schematics and PCB of COMBO6 card. Technical Report 14/2002, CESNET, 2002.

www.icgst.com

Reconfigurable Image Processing Architecture

P. Zemčík, A. Herout, V. Beran
Department of Computer Graphics and Multimedia, Faculty of Information Technology
Brno University of Technology, Božetěchova 2, CZ-612 66 Brno, Czech Republic
[zemcik, herout, beranv]@fit.vutbr.cz
http://www.fit.vutbr.cz

O. Fučík
Department of Computer Systems and Networks, Faculty of Information Technology
Brno University of Technology, Božetěchova 2, CZ-612 66 Brno, Czech Republic
fucik@fit.vutbr.cz
http://www.fit.vutbr.cz

J. Schier
Institute of Information Theory and Automation, Czech Academy of Sciences
Pod vodárenskou věží 4, P.O. BOX 18, 182 08 Praha 8, Czech Republic
schier@utia.cas.cz
http://www.utia.cas.cz

Abstract
This paper presents a novel concept of image processing architecture based on interconnection of the programmable logical chips (FPGAs) and digital signal processors (DSPs). Each of these technologies has very good features for some classes of image processing tasks while they fail to efficiently execute tasks of other classes. As these classes are different and nearly complementary for the above mentioned technologies, the combination of such technologies may achieve good results in larger classes of tasks. This paper also presents methods for development of applications for the proposed architecture and summarizes the potential of the architecture.

Keywords: *Image processing, Digital signal processing, Programmable Logic, FPGA.*

1. Introduction
Image processing generally exploits tasks with very high computational demands. Such tasks can be handled by the "standard" processors and computers or by networks of such computers; however, such solution is not always feasible for various reasons, such as difficulties with programming in multiprocessor system, large dimensions of the computer system, high consumption of energy, etc.
An alternative method of handling the high computational power demands and specific image processing requirements is usage of specialized processors, such as digital signal processors (DSPs), programmable digital circuits, such as field programmable gate arrays (FPGAs), or combination of both, which is proposed in this paper.
The strong point of the DSPs is in their specialized architecture focused on multiply and accumulate operations (MACs) in which the current DSPs (e.g. Texas Instruments C64 architecture [9]) often outperform the "standard" processors. The DSPs at the same time have low energy consumption, easy to use architecture, and other nice hardware features; however, DSPs still suffer from the disadvantages of all the sequential processors, such as lack of massively parallel data processing, difficult bit manipulation, fixed data width, etc.
FPGAs, on the other hand, have a large potential of performing many operations simultaneously, such as addition, multiplication, logical operations, etc. However, FPGAs in general suffer from relatively small on-chip memory capacity and also from relatively narrow bandwidth memory interfaces, lack of wide-word processing units, and also unavailability of units capable of performing complex numerical operations, such as division, square root, logarithmic, exponential, and goniometrical functions. Also, complex memory controllers and addressing units are difficult and expensive to implement.
Combination of DSPs (or other sequential processors) and FPGAs (or other programmable logical circuits) are subject of research for several years already [6,7,8,2,4,5]. Although the features of the above mentioned two technologies are suitable for combination, development of applications for such combinations is generally

difficult and really applicable automated tools for distribution of the computational tasks between the processors and programmable logic are generally not available yet; therefore, the application development support tools and methodology are probably as important as the potential of the architecture combining the processors and programmable logic. This paper attempts to address both of the issues.

2. Architecture

To achieve high performance in image processing tasks, state-of-the-art components must be used. In the presented architecture, high performance Texas Instruments C64xx series DSPs [9] were used along with the powerful Xilinx Virtes II FPGAs [10]. These devices are currently among the most advanced ones available on the market; however, the architecture of the system and the development methods are generally applicable to virtually any similar architecture.

The proposed architecture consists of a miniature "core computational module". One or more of such modules can be used in various applications and depending on the application the module(s) should be interconnected with suitable interfacing circuits. This paper presents an example of such setup with four computational core modules placed on a PCI "motherboard" carrying the modules and providing their mutual interconnection, PCI interface, and a number of interface pins available for application-specific optional module.

The design of the set of modules has been done jointly with CAMEA, spol. s r.o. (ltd.) who also manufactures all the modules. The current core computational module, DX64, had two successful predecessors, DSPX10 (based on Texas Instruments C32 series DSP and Xilinx Spartan FPGA) and DX6 (based on Texas Instruments C67 DSPs and Xilinx Virtex-E FPGAs) [3,1].

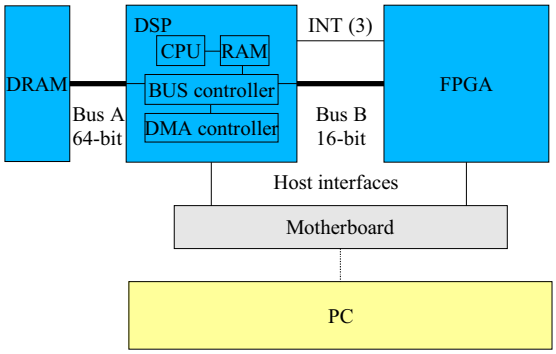


Fig. 1: Block diagram of the computational module.

To exploit the features of the FPGA and DSP in the best feasible way, it was decided that the FPGA will mostly rely on data transfers and data storage provided by the DSP. The FPGA is, therefore, connected to the peripheral bus interface of the DSP and is accessible as a "set of registers" in the DSP's memory space. Such design has been successful on the predecessors of the current design. The C64xx series of DSPs have two external buses dedicated for memory interface and peripheral interface. The FPGA is connected to the peripheral bus so it does not affect the memory bandwidth of the system and the DSP's raw computational speed. The data delivery to and from the FPGA is accomplished through the DMA controllers built in the DSP. See Fig. 1 for the block diagram.

Physically, the module is a small board with surface mounted electronic components. The module is thin to allow for itself and the motherboard to occupy only one PCI slot. The photograph of the system can be seen in Fig. 2.

Fig. 2: Photograph of the core computational module

The motherboard to carry the four core computational modules contains merely a PCI interface, additional memory controller unit, and expansion and interface circuits. These circuits are also built using FPGA chips so that interfacing of the modules to the motherboard is not too complex. The block diagram can be seen in Fig. 3.

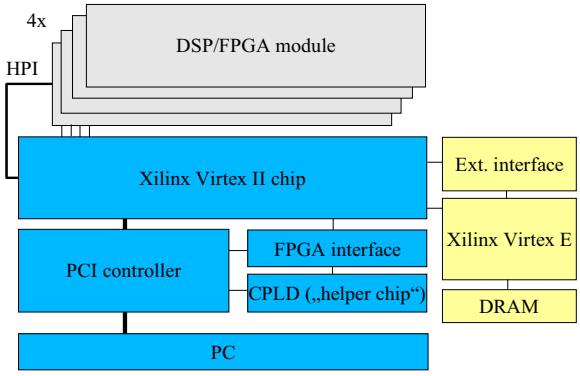


Fig. 3: Motherboard block diagram

The photograph of the actual motherboard can be seen in the Fig. 4. The physical layout of the board is a "full size PCI board" that occupies one PCI slot. As the power consumption of the whole design is relatively small (around 25W) the host computer system can carry multiple motherboards simultaneously. The motherboards can be interconnected though a Low Voltage Differential Signaling (LVDS) interface that is automatically contained in the FPGAs used on the board.

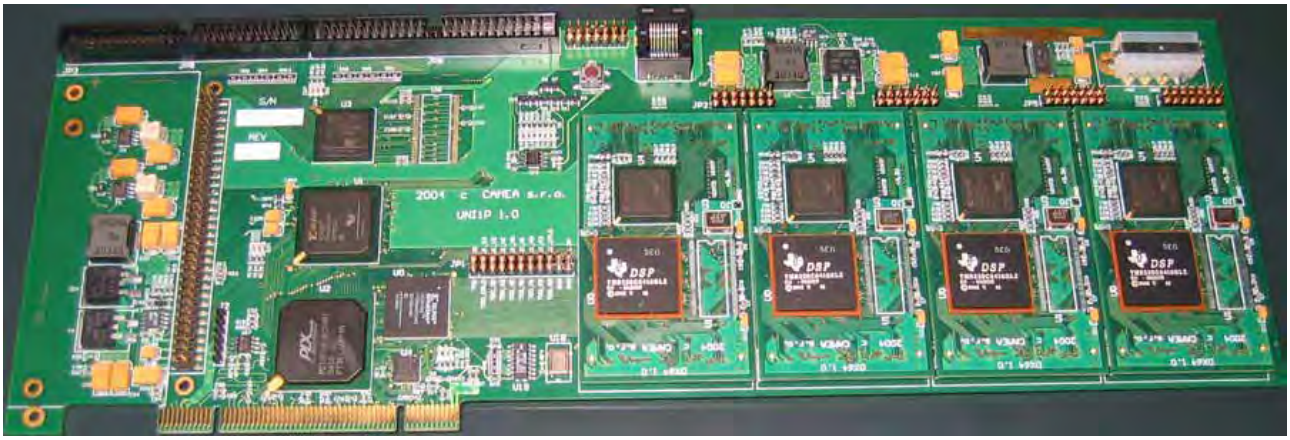


Fig. 4: Photograph of the motherboard carrying four computational modules

The basic setup of the motherboard and of the modules is done through a set of "C" functions that are available in UNIX (Linux) and Windows version. These functions provide means of PCI configuration, FPGA design upload, DSP software upload, etc. While these functions are specialized for the motherboard, the FPGA designs and DPS software they upload are generic and can be used in any configuration of the core computational modules.

3. Application Design

The crucial task in design of the image processing architectures like the one presented in this paper is to address the application design process. The "standard" approach to the application design would be to use the tools provided by the component manufacturers, such as Texas Instruments Code Composer Studio for the DSPs and Xilinx VHDL development tools for the FPGAs. Unfortunately, very little of the real-life image processing application designers are familiar with both of these tools. Additionally, the tools are rather expensive and difficult to use. Moreover, the application designers are not used to such tools at all.

The idea of application development on the presented platform is to allow the designers to keep the model of the application similar to what they are used to in standard computer programming. The algorithms are encoded as single or multi-threaded pieces of software using some kind of a procedural notation. The block diagram of the software development modules relationship is shown in Fig. 5.

The "standard" method of coding the application is in the Perl script language or alternatively in Parrot "assembly language-like" intermediate code. Perl/Parrot code is compiled into a binary (byte-stream) form that is then executed in the DSP/FPGA system or in a PC. The binary code is accompanied with the function library which is implemented in "C" language and some of its functions (the computationally demanding ones) in VHDL in order to be placed in the FPGA. Note, please, that while the application development is performed in Perl, the critical functions are written in "C" and optionally in VHDL.

In some cases, the applications do not have to be described in Perl language just through the Editor/Debugger environment but they can be automatically generated. An example can be automatic block diagram transfer from Simulink into Perl using the Simulink TLC (Target Language Compiler tool) that can convert the block diagram to virtually any procedural language (Perl as well). For some user-specific cases, specialized applications can be prepared for PC platform in order to allow the users to create applications e.g. through graphical interfaces. The applications are then automatically converted into Perl script (or directly into Parrot) and transferred into the target platform, potentially without the user knowing about the intermediate steps and about the script at all.

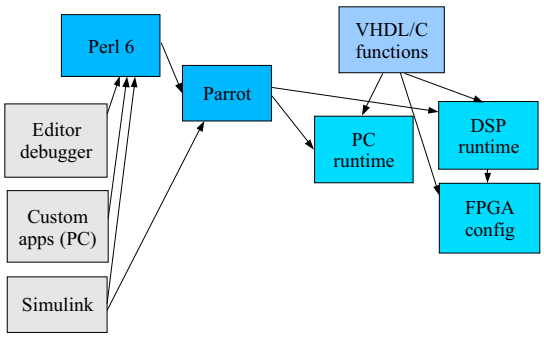


Fig. 5: Software development modules

While the compiler available for the target DSP platform by the manufacturer is a C/C++ compiler, as mentioned above, it was decided that a better platform for application development would be an open source scripting language (or language compiled in some form of a "byte-stream") rather than a commercial "C" compiler. The main reasons are the following:

- The development process using a scripting language is smoother and easy to handle,

- the development tools for "end users" can be made open source which expands the set of potential users,
- the runtime can be changed without linking the target application (like e.g. DLLs on conventional platforms),
- if the scripting language is carefully selected, the execution can be nearly as efficient as with compiled code,
- the script compiler can be incorporated into high-level debugging tools.

Selection of the script language to use was rather difficult. Several rather widely spread languages were considered and found not useable for our purposes. The reasons were for example the following:

- JAVA – the main problems with JAVA is its lack of efficient open source (or DSP platform) runtime, too extensive standard libraries, and garbage collecting memory management approach.
- JAVAScript – similar to JAVA, specifically lack of efficient open source compiler and runtime was the main problem.
- FORTH – the problem with this language was that although it is simple and open source is available, it is inefficient.

Finally, Perl was selected as the suitable choice for its open source nature, rather wide penetration, and also an efficient intermediate "assembly language like" code (Parrot, available in Perl version 6) whose runtime engine is efficient and also possible to modify in order to incorporate possible extensions of functionality in image processing, multithread application synchronization, etc.

4. Image Processing in FPGA

Implementation of the image processing functions if the FPGA is done through a set of pre-defined blocks with various functions. These blocks correspond to the pre-defined "C" and VHDL functions in the above mentioned library of functions. The blocks in the FPGA are interconnected so that they perform the operations defined by the script. Each block's output can be connected to a set of other blocks' inputs – this means 1:N interconnection; however, the implementation of the interconnection is partially sequential, not fully parallel. Each of the blocks can have arbitrary number of inputs and outputs depending on its functionality. The input and output of the data to and from the DSP is also treated inputs and outputs similar to the inputs and outputs of the functional blocks. DSP interface contains merely a simple control logic and buffers. The DSP interface block can also have arbitrary number of inputs and outputs and they correspond to DSPs' DMA controllers. The communication "wires" are 16-bit words.

The configuration of the FPGA can be static – the set of function blocks can be adjusted to the actual application or class of applications – or dynamic – the set of function blocks can be exchanged "on the fly" through FPGA dynamic reconfiguration under the control of the DSP runtime. See Fig. 6 for the example of the FPGA structure.

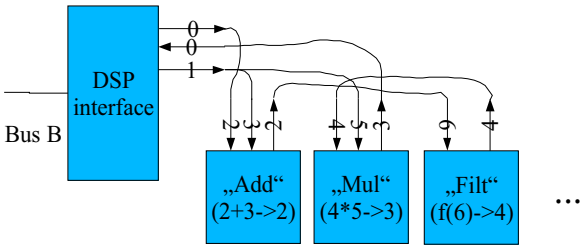


Fig. 6: FPGA image processing structure

The FPGA structures available for the functional blocks are apart of the standard FPGA structures (logical functions and D-flip-flops) also small SRAM blocks (24-102 blocks with 2KB size each in this case) and 18x18 bit fast multipliers (24-102 multipliers). Optionally, one of the function blocks can use the DRAM interface provided on the board for a necessary local storage (local image buffers, FIR coefficient buffers, etc.).

5. Design Example

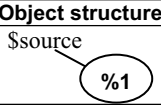
A trivial example of the process of interpretation of a script, its translation to a sequence of calls to the underlying layer and the final execution of the operations is in Fig. 7. This figure shows a very simplistic image-processing sample code which firstly sets up an image source (can be a digital line connected to the presented system, an extension module for grabbing analog video signal, or other means of input) and a convolution mask. Afterwards, an infinite loop gets an image, convolves it with the mask, cuts out a region of interest and runs a function to find peaks in the convolved image. In case the peak is high enough, the original image is sent for further processing into another part of the system. The script code solving this problem is shown in Fig. 7.

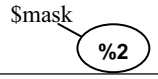
```
$source = new ImageSource("src1");
$mask = new ConvMask( 1,  2,  1,
                      0,  0,  0,
                     -1, -2, -1);
for (;;) {
  $img = $source->GetImage();
  $clip = cut(($img * $mask),
              50, 10, 200, 180);
  ($x, $y, $val) = findpeak($clip);
  if ($val > 50) {
    SendMessage($x, $y, $clip);
  }
}
```

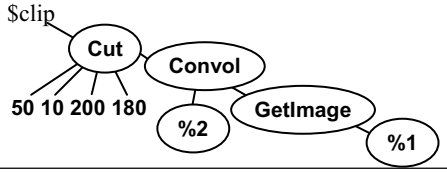
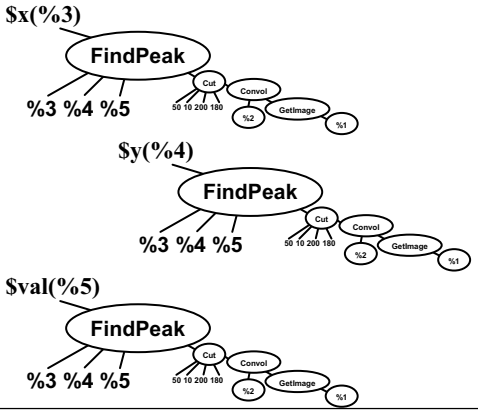
Fig. 7: Scripted code of the processing sample

The left column of the table cites a line in the scripting code, which is the program written by the ending user of the presented application framework. The code operates on objects (in the sense of Object-Oriented Programming methodology) which by calling their methods explicitly (as in "$source->GetImage();") or implicitly (as the implicit conversion in "if ($val > 50) {") create and

dispose structures of objects, arranged into n-ary trees or more general directed acyclic graph structures. Relevant portions of these structures are illustrated by the second column of the table. The third column of the table in Fig. 8 shows symbolic formulations of functional calls that the scripted code makes to the underlying execution layer. The execution environment performs optimizations and simplifications of the tree-like structure it operates on and finally performs the actions using the hardware resources on the acceleration board.

Script code	Object structure	Calls to underlying layer
`$source = new ImageSource("src1");`		"%1 newsource"

`$mask = new ConvMask( 1, 2, 1,`
`0, 0, 0,`
`-1, -2, -1);`
		"1 2 1 0 0 0 -1 -2 -1
		%2 newconvmask"

`for (;;) {` `$img = $source->GetImage();`		
`$clip = cut(($img * $mask),` `50, 10, 200, 180);`		
`($x, $y, $val) = FindPeak($clip);`		
`if ($val > 50) {`		"%1 GetImage %2 Convol 50 10 200 180 Cut %3 %4 %5 FindPeak" "%5 GetNum"

`SendMessage($x, $y, $img);`
`}`
`}`
		"%3 GetNum"
		"%4 GetNum"
		"%1 GetImage"

Fig. 8: Analysis of the processing sample

Several methods of optimizing DAGs (directed acyclic graph) are or will be imported from the theory and practice of compilers and code optimization. In the above example, the operation of cutting can be performed before the convolution (in defiance of the literal meaning of the script code), thus greatly reducing the number of pixels processed (See Fig. 9).

Please note in Fig. 8 that the calls to the execution environment are performed very scarcely, with DAG structures as spreading as possible to both decrease the communication bandwidth and allow the execution environment the widest possible space for optimizations.

Complex operations are thus composed (as the one shown in Fig. 9), which from the point of view of the user-level scripting tool appear atomic and can be therefore implemented by a pipeline in the hardware accelerator without the need of transferring the intermediate phases to/from buffer memory or the host processor interpreting the script.

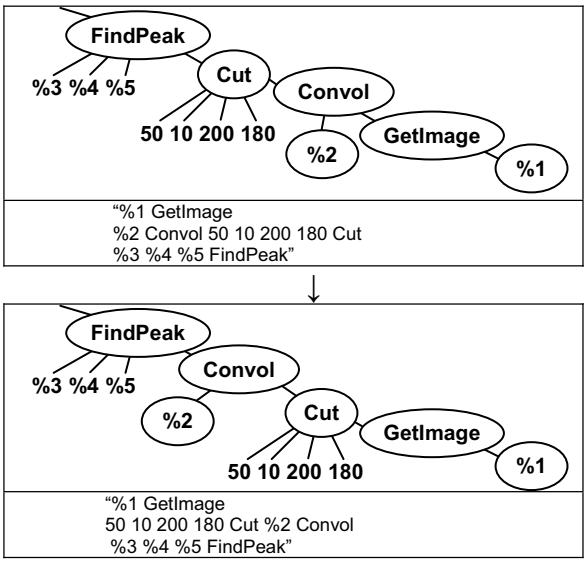


Fig. 9: An example of optimization

Just as in the simple example given here, in real image-processing applications large portions of the DAG structures tend to repeat between execution cycles performed repeatedly (typically in a virtually infinite loop). These repeating portions of the tree can be identified either by a simulation in the system-design phase or in run-time, and will be pre-set to the acceleration hardware, thus mimicking the mechanism of just-in-time compiling.

6. Design Potential

The performance of the presented system hardware cannot be exactly determined without specifying the application; however, the computational performance (already implemented or projected) of the examples of system modules is shown in the following table (please, note that the modules run in parallel):

Module	Description	Performance
DSP C6411	Currently available in 1GHz version, up to 16 instructions in parallel (VLIW architecture)	Up to 4 billion MAC/s
Constant multiplier, adder in FPGA	Running at least at 50 MHz, FPGA can contain arbitrary number of these units depending on its size	50 million pixels/s
Multiplier in FPGA	Running at least at 50 MHz, FPGA can contain 24-102 multiplier units depending on its size	50 million pixels/s
Non-linear tabulated function	Running at least at 25 MHz, FPGA can contain 24-102 non-linear units depending on its size (contains SRAM)	25 million pixels/s
Convolution filter up to 5x5 size	Running at least at 25 MHz, FPGA can contain arbitrary number of these units depending on its size	25 million pixels/s
Median or rank-order filter up to 5x5 size	Running at least at 25 MHz, FPGA can contain arbitrary number of these units depending on its size	25 million pixels/s
Linear classifier up to 32x32 size	Running at least at 50 MHz, FPGA can contain 24-102 multiplier units depending on its size (contains multiplier and SRAM), the unit contains 8 parallel classifiers	100 million features/s
Statistics collection (histograms etc.)	Running at least at 50 MHz, FPGA can contain 24-102 multiplier units depending on its size (can contain SRAM)	100 million features/s

7. Conclusions

The presented system demonstrates the potential of the reconfigurable image processing architecture based on the combination of the DSPs and FPGAs for raster image processing. One of the major obstacles in usage of such systems – complicated application development – is addressed as well and the proposed solution is to use a scripting language for overall application description and "C" language and VHDL to code the library image processing functions. Although such approach is, of course, not ideal, it can be seen as an immediately useable approach that allows for application development at the current state of the art in computing.

The hardware architecture presented in this paper is, from the purely computational speed point of view, very powerful architecture for image processing that, at the time of writing the paper, several times faster comparing to a standard PC in the terms of MAC per second (the actual speedup factor varies very much depending on the application).

Future work on the architecture includes hardware improvements to reflect the state of the art in DSP and FPGA, further development of the tools, and research in the image processing methods in order to allow efficient implementation in the DSP/FPGA environment that can often lead to complete restructuring and modification of "traditional" algorithms.

8. Acknowledgments

The authors of this paper would like to thank CAMEA, spol. s r.o., namely Mr. Pavel Tupec and to Mr. Radek Bardas and Antonin Hegar for their help in implementation of the system. The research was funded and thanks also go to the Grant Agency of the Czech Academy of Sciences for the grant "Rapid prototyping tools for development of HW-accelerated embedded image- and video-processing applications", GA AVČR, T400750408.

9. References

[1] Herout A., Zemčík P., Beran V., Kadlec J.: Image and Video Processing Software Framework for Fast Application Development, In: AMI/PASCAL/IM2/M4 workshop, Martigny, CH, 2004, p. 1

[2] Zemčík P., Herout A., Crha L., Tupec P., Fučík O.: Particle rendering pipeline in DSP and FPGA, In: Proceedings of Engineering of Computer-Based Systems, Los Alamitos, US, IEEE CS, 2004, p. 361-368, ISBN 0-7695-2125-8

[3] Crha L., Fučík O., Zemčík P., Drábek V., Tupec P.: Inter Chip Communicating System with Dynamically Reconfigurable Hadrware Support, Poznaň, PL, 2003, p. 311-312, ISBN 83-7143-557-6

[4] Tišnovský P., Herout A., Zemčík P.: Cache-Based Parallel Particle Rendering Engine, In: ElectronicsLetters.com, Vol. 2003, No. 1, Brno, CZ, p. 8, ISSN 1213-161X

[5] Zemčík P., Herout A., Tišnovský P.: Particle Rendering Pipeline, In: Spring Conference on Computer Graphics 2003 Proceedings, Bratislava, SK, STUBA, 2003, p. 180-186, ISBN 80-223-1837-X

[6] Fučík O., Zemčík P.: Hardware Accelerated Imaging Algorithms, In: AUTOS 2002 Automatizace systémů, Praha, CZ, ČVUT, 2002, p. 165-171, ISBN 1213-8134

[7] Zemčík P.: Hardware Acceleration of Graphics and Imaging Algorithms using FPGAs, In: Proceedings of SCCG, Budmerice, SK, STUBA, 2002, p. 8

[8] Zemčík P., Fučík O., Richter M., Valenta P.: Imaging Algorithm Speedup Using Co-Design, In: Summaries Volume Process Control 01, Štrbské Pleso, SK, 2001, p. 96-97, ISBN 80-227-1542-5

[9] TMS3B0C6711, TMS320C6711B Floating point Digital Signal Processors, Texas Instruments, SPRS088B, USA, (available at http://www.ti.com)

[10] Virtex™-E 1.8 V Extended Memory Field Programmable Gate Arrays, Xilinx, DS025-2 (v2.2), USA, (available at http://www.xilinx.com)

www.icgst.com

Image Processing in Traffic Applications

P. Zemčík, A. Herout, V. Beran, I. Potúček
Department of Computer Graphics and Multimedia,
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, CZ-612 66 Brno, Czech Republic
[zemcik, herout, beranv, potucek]@fit.vutbr.cz,
http://www.fit.vutbr.cz/units/UPGM/

O. Fučík
Department of Computer Systems and Networks,
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, CZ-612 66 Brno, Czech Republic
[fucik]@fit.vutbr.cz,
http://www.fit.vutbr.cz/units/UPSY/

J. Honec, M. Richter, I. Kalová, M. Lisztwan
Department of Control and Instrumentation,
Faculty of Electrical Engineering and Communication, Brno University of Technology
Božetěchova 2, CZ-612 66 Brno, Czech Republic
[honec, richter, kalova, lisztwan]@dame.feec. vutbr.cz,
http://www.feec.vutbr.cz

Abstract
This paper presents overall information about the Unicam traffic enforcement and monitoring system based on image processing. The system uses a set of spatially distributed sensor and video acquisition units interconnected using a computer network that allows efficient data transfers and exchange of data between the functional units. The system also allows remote networked access to the data and results of the processing and allows for automated data transfer and interconnection of the subsystems to a host system. In this paper, the overall structure of the system is given and the main image processing algorithms used are roughly described.

Keywords: *Traffic surveillance, Image processing, Networked system.*

1. Introduction
Automotive traffic is both very important and very problematic issue in most of the developed countries around the world. Traffic control and enforcement are generally accepted as suitable methods to handle the increasing density of traffic. While the technology of sensors, electronics, etc. is constantly improving, severe limits exist for exploiting such technologies on the real roads. The limitation lies in both technical and legal complications in installation of the physically connected sensors to the road – the roads would have to be modified in order to connect the sensors which stop the traffic and often the construction of sensors must be approved before use. Therefore, non-contact technologies, such as microwave systems, laser-based systems, and also passive image processing systems are attractive for traffic applications. The microwave systems, especially Doppler speed-enforcement radars are popular for years. Nowadays, as the computer technology is constantly more and more capable of handling often difficult computational tasks, computer vision technology and its advanced methods allow for creation of passive non-contact applications based on video processing that could provide extra information about the traffic and individual vehicles [1][2][3].
This paper presents an overview of complex traffic application system that integrates several functions of the video processing in traffic into a complex networked system. The technology described in this paper has been developed in a close co-operation with CAMEA, spol. s r.o. (Brno, Czech Republic); this company also manufactures most of the equipment used in the system.

2. System Functions
The general goal of the traffic image processing system is to provide suitable functionality and allow integration into a larger structure – it would be naive to think that a single solution would cover all the needs of traffic

authorities. Modular systems are, therefore, the only reasonable approach.

The core image processing functionality of the described system includes the following basic functions:

- detection of cars on the road "checkpoints"
- acquisition, storage and precise time stamping of the images/video
- recognition of the vehicle class
- detection of the traffic-lights signaling
- recognition of the vehicle registration plate
- matching of vehicle images or vehicle part images

These functions can be exploited in higher-level functions, such as:

- *red light enforcement* – combination of the car detection, detection of the traffic-lights signaling, and storage
- *speed enforcement* – combination of car detection, matching of the vehicle images, and storage
- *toll-collection enforcement* – recognition of vehicle class combined with car detection
- *traffic monitoring and vehicle tracking* – recognition of vehicles and vehicle classes in combination with storage

The following sections contain a more detailed description of the solutions used in the modules of the system.

3. Vehicle Detection Sensors

The car detector is based on an active triangulation technique. The light source (laser), the detector (line CCD camera) and the illuminated part of the measured object form a triangulation triangle. The light source ray angle is fixed whereas the angle on the detector side changes. Based on the knowledge of two angles and one side of the triangle, the distance can be determined. The system, formed from the camera and lasers fixed on a platform, is placed above the roadway. (See Fig. 1.) The measurement rate is given by the sampling frequency of the camera (around 2ms/sample). The presence and height profile of the vehicle is acquired in this way. Multiple lasers are used to provide sufficient amount of data for measurement e.g. on the whole traffic lane width. The rough velocity of the vehicle can also be obtained by using a second system situated behind the first in a defined distance and by calculating the time shift between the two profiles. The profile can also be used for vehicle class recognition which is described further.

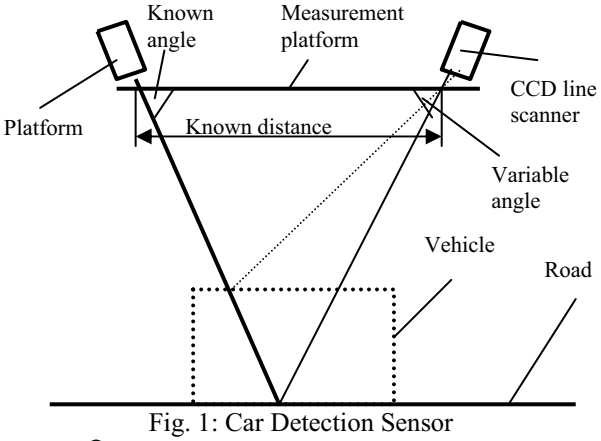


Fig. 1: Car Detection Sensor

Another technique of detecting cars used in the presented system is based merely on video data passively grabbed in HDTV resolution from CCD cameras (contained in the further described video acquire units). The vehicles passing through the field of view of the camera are preliminarily detected by a combination of simple and fast frequency filters and a clustering algorithm. These preliminary detections are verified and either confirmed or dropped by a subsequent complex algorithm operating on frequency transformed data.

4. Video Acquisition Units

For acquiring video data, CCD cameras coupled with a image-processing board DX6 featuring a TI C6711 DSP [4] and a Xilinx Virtex E-300 FPGA (Field-Programmable Gate Array) [5] are used. The unit provides extra functionality to simple grabbing and transmitting of video signal. Preliminary visual detection of cars (as referred to in the previous section) is fully implemented in these units [6][7], accurate (GPS synchronized) timestamps are also provided to allow high precision of time information necessary for vehicle speed measurement in the overall distributed system. The unit is capable of communication through wired (100 MBit/s Ethernet) or wireless (GPRS or CDMA) network communication. See Fig. 2 for the actual system photograph.

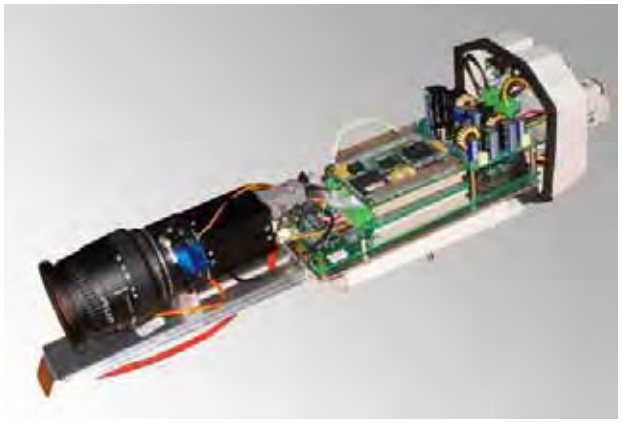


Fig. 2: Intelligent camera featuring a TI DSP C6711 and FPGA Xilinx E300

5. Vehicle Class Recognition

The vehicle class recognition is based upon classification of vehicle profiles acquired through the above mentioned laser scanning system. The height profile is derived by image processing methods executed on digital images coming from high speed and high resolution line digital camera. From every single acquired profile a set of attributes is computed which is compared to the attributes of the "ideal" profile of the class representative stored in a database. The volume of the database depends on required quality of classification, number of classes to which the classification is performed and respective speed regards. The selection of best fitting class is executed on the set of profiles with the highest score. Resulting vehicle class such as "bus", "truck mixer", "pickup" etc., could be easily transformed to a more useful information for example vehicle typical weight,

number of axles and so on. See Fig. 3 for examples of scanned profiles and respective vehicles.

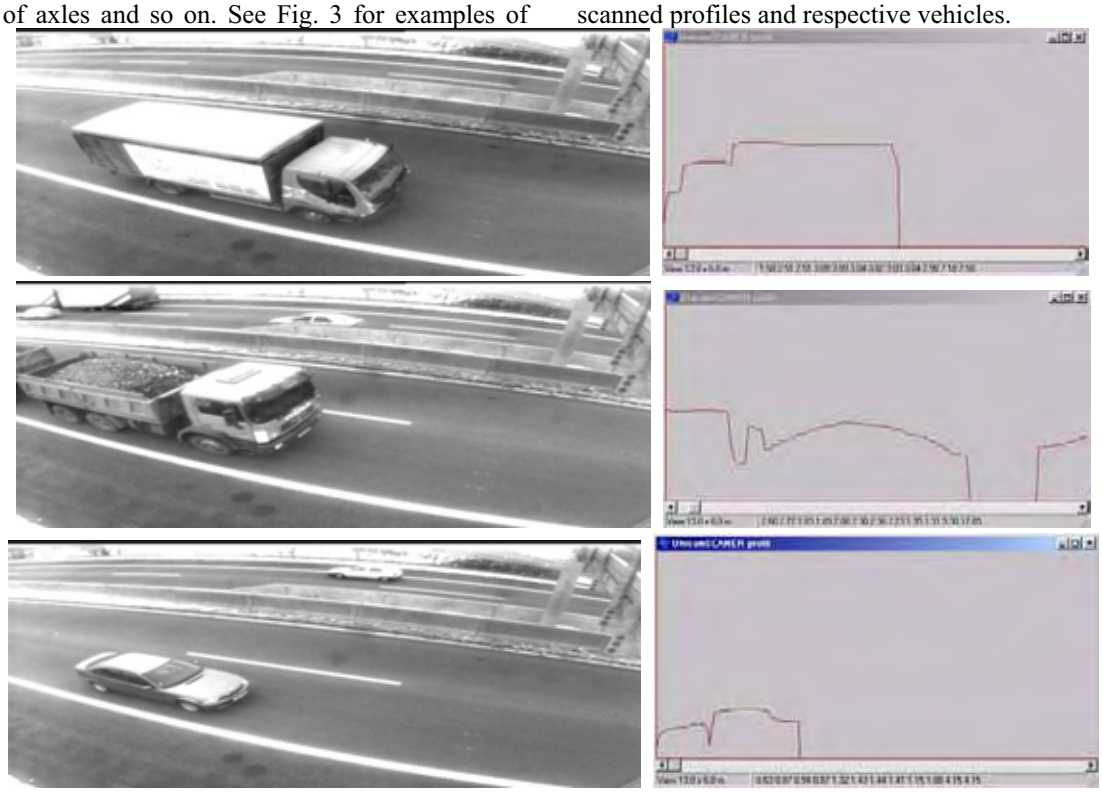


Fig. 3: Vehicle class recognition from vertical profiles scanned by the profile scanner

6. Traffic Lights Signalling

The traffic lights signalling is an important source of information in the system. In the theory, the phases signalling provided by the traffic lights can be available from the various traffic lights controllers. Such approach is simple but in practice two problems occur:

- The manufacturers of the traffic lights signalling systems are not too keen to open the controllers for the traffic monitoring and enforcement systems they do not manufacture for both commercial and traffic safety reasons.
- In some cases, e.g. if the light bulbs are failing, if the wiring fails, or if some obstacles occlude the traffic lights, the drivers cannot be aware of e.g. the red light. Video processing, in this case, provides both the information about the traffic light phase and the evidence that the driver was able to see the actual light.

Image processing methods used for the purpose of light phase recognition are relatively simple as a fixed video camera setup is used so just traffic light position refinement and lightness-based methods are used.

7. Registration Plate and Vehicle Matching

The recognized registration plate or images of the vehicle number plate are used for vehicle matching. A third-party algorithm is used for registration plate recognition. The algorithms based on the recognized registration plate (the text string) can be easily used in a distributed environment as the amount of data that has to be shared among the video acquire units and the computers is small (several bytes per car). Specifically, if the video acquire units are distant one from another and no wired network is available, usage of the wireless (GPRS or CDMA) network and this is the only option.

The other possibility is to directly compare the couples of images of the vehicle registration plates and based on the image comparison to decide whether they belong to the same car or not (without necessarily knowing the content of the plate). Obviously, this approach is more robust than the registration plate recognition. Especially the image comparison approach can use registration plates whose format is unknown (e.g. foreign plates); however, this approach is much more costly both from the point of view of communication (registration plate image contains about 2KB of data) and processing power comparison of the incoming image to the whole set of candidates must be done (no ordering is possible). The algorithms used for image-based matching of images of vehicles basically divide the license plate into regions of interest – often (but not necessarily) corresponding to separate characters of the license plate – and test them against regions selected from a set of potentially correspondent images. Careful attention is paid to equalization of light conditions from one image to the other, since the locations of different cameras can be exposed to very different measures of sunlight and shadow – see Fig. 4.

Fig. 4: Couple of images of a single car to be matched

The presented system combines both of the above mentioned approaches and exploits the knowledge about paired couples of images for vehicle tracking, traffic data acquisition, and for speed measurement (along with the information about the precise location of the sensors and timing of the images).

8. Networked Data Processing

The data acquired by the sensors and video acquisition units need to be further processed. The nature of the system is such that the spatial localization of the sensors and the image acquire units is distributed and that the mutual distances of the systems are relatively large (from units of meters to thousands of meters).

As indicated above, the data from various localities need to be integrated for the purpose of gathering overall traffic statistics and all crucial data must be available to various traffic authorities either online or in separate batches carried on high-capacity media. To ensure all the versatile means of interconnection, networking of different scales is involved in the system. The first generation of the design is illustrated by Fig. 5.

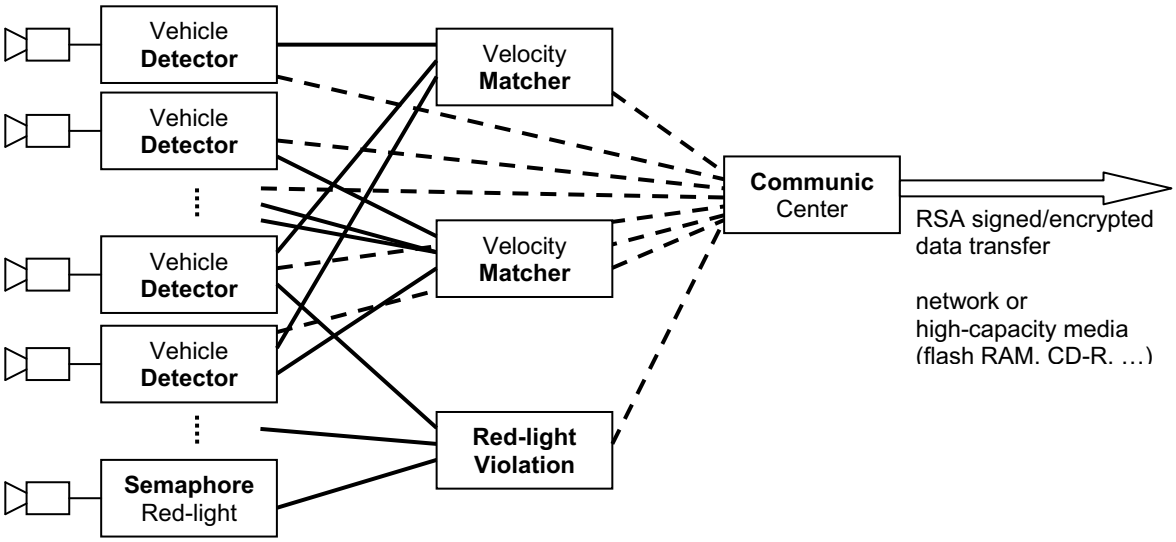


Fig. 5: Tightly connected variant of the system in one locality

Wired connections within one locality (as shown in Fig. 5) allow relatively simple multiprocessing within one local installation of the system, since the connections can be considered reliable and provide sufficient bandwidth to communicate all necessary data in real-time. However, as mentioned above, the need for wired connection can be very limiting in some cases, because mutual distances between separate systems and even small subsystems and detection devices within one system can extend to hundreds or thousands of meters. This fact leads to the (more complex) solution based on different communication technologies as shown in Fig. 6. This solution supports both wired and wireless communication [8][9].

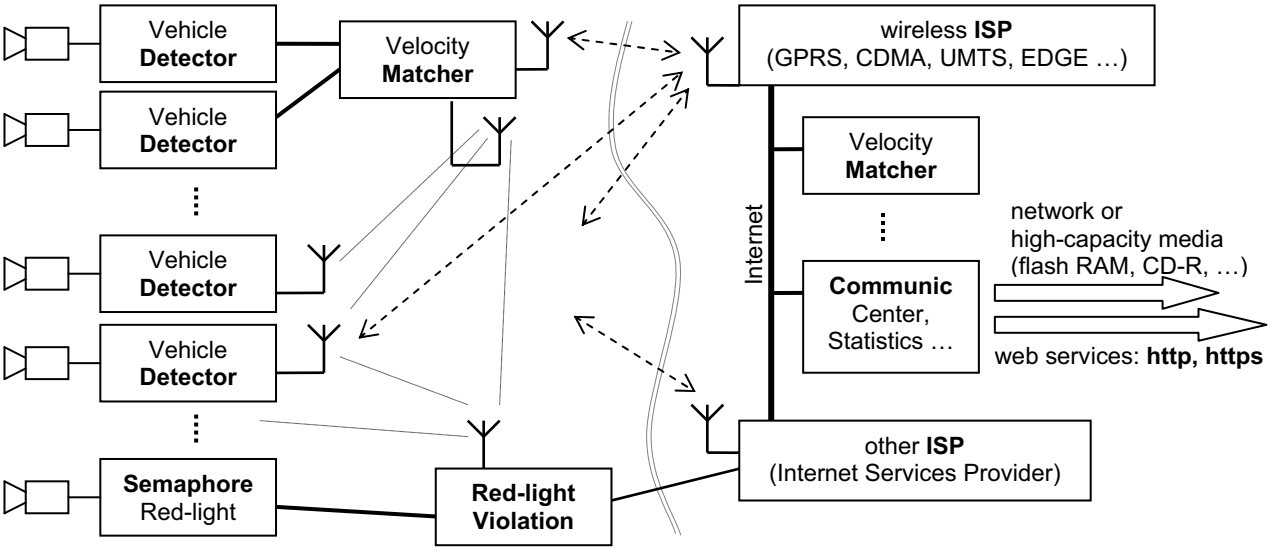


Fig. 6: Wireless communication enabling system

Wireless connections (Wi-Fi for short ranges within one locality, mobile technologies for wide distances) allow some simplifications to the system, especially the ability to locate some coplex parts of the system indoor and on-line sharing of the data necessary for instant overall statistics. Also the fact that the system is intrinsically attached to high-speed Internet allows utilization of well established internet technologies (sql, https, ssh) for sharing the data with the end users as well as remote control and maintenance of the image processing (and other) sensors displaced in the field.

These important benefits of wireless communication and overall integration of the system were paid for by increased need for security in the system and other complications avoided by a closed-circuit distributed system. The wireless communication cannot be considered reliable but just the opposite – the system must assume failures lasting even hours and the bandwidth of the communication channel is far from sufficient to handle peak loads. A new need for high-capacity non-volatile buffers (capable of storing hundreds to thousands of HDTV images in each wireless-connected image processing sensor) was therefore raised. Also the latencies of the communication play a painful role in the process of gathering proper offence documentation, which must include snapshots from different spatial and time locations around the moment when the offence occurred – the system needed to be restructured to cope with these requirements, imposing further needs for image buffers in all involved nodes. Another problematic issue faced in this loosely-coupled version of the presented distributed image-processing system was time synchronization which is finally solved by GPS stations attached to single sensors or clusters of sensors closely connected.

9. Applications

The system presented in this paper has been up till now applied in a number of real-life applications. The applications include red-light enforcement systems (e.g. over 100 measurement points in Prague, Czech Republic), speed enforcement (e.g. 3 locations in Prague, Czech Republic with over 20 measurement points), traffic statistics collection (working experimental installation in Brno, Czech Republic), and vehicle tracking and stolen vehicle detection (experimental systems running in Brno, Czech Republic).

The systems have mostly web front-end adjusted to the customers' needs. The demonstration of the user interfaces can be seen in Fig. 7.

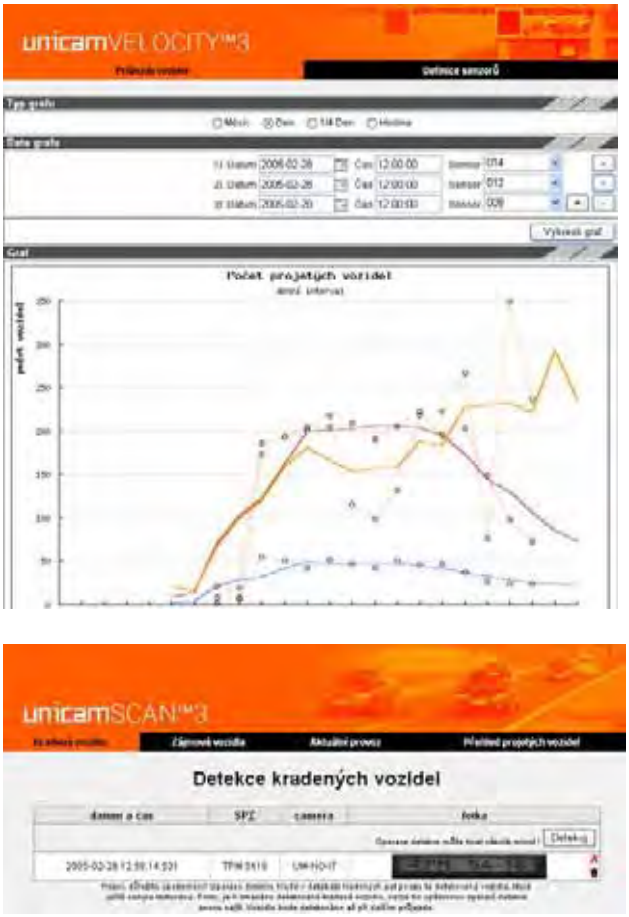


Fig. 7: Stolen vehicle and traffic statistics web front-ends

10. Conclusion

The presented traffic image processing system has been successfully implemented, tested, and applied in the field. While the development of the system is still in progress and the functionality of the system constantly improves, it has been shown that the basic concept of the system is feasible and that this class of systems is applicable.

Future work on the system includes development of battery-powered standalone video acquisition units and sensors, better algorithms for car detection, robust registration plate recognition, real-time vehicle tracking and recognition, and other algorithms for new system functionality.

11. Acknowledgements

The authors would like to thank CAMEA, spol. s r.o., namely Mr. Tomáš Bia and Pavel Valenta for the co-operation in implementation of the system. The research was funded and thanks also go to the Grant Agency of the Czech Academy of Sciences for the grant “Rapid prototyping tools for development of HW-accelerated embedded image- and video-processing applications”, GA AVČR, T400750408.

12. References

[1] Jerry V. Capozzi and Stephen L. Benning, *"Multi-Lane Traffic Monitoring System (MTMS)"*, 1998 SAE FTT Future Transportation Technology Conference and Exposition, USA, June 1998

[2] Asgari H., Trimintzios P., Irons M., Egan R., Pavlou G.: *Building Quality of Service Monitoring Systems for Traffic Engineering and Service Management*, Journal of Network and System Management, Vol. 11, No. 4, pp. 399-426, Plenum Publishing, UK, December 2003

[3] Fučík O., Zemčík P., Tupec P., Crha L., Herout A.: *The Networked Photo-Enforcement and Traffic Monitoring System*, In: Proceedings of Engineering of Computer-Based Systems, Los Alamitos, US, IEEE CS, 2004, p. 423-428, ISBN 0-7695-2125-8

[4] *TMS3B0C6711, TMS320C6711B Floating point Digital Signal Processors*, Texas Instruments, SPRS088B, USA, September 2001, (available at http://www.ti.com)

[5] *Virtex™-E 1.8 V Extended Memory Field Programmable Gate Arrays*, Xilinx, DS025-2 (v2.2), USA, September 2002, (available at http://www.xilinx.com)

[6] Zemčík P.: *Hardware Acceleration of Graphics and Imaging Algorithms using FPGAs*, In: Proceedings of SCCG, Budmerice, SK, STUBA, 2002, p. 8

[7] Zemčík P., Richter M., Valenta P.: *Imaging Algorithm Speedup Using Co-Design*, In: Summaries Volume Process Control 01, Štrbské Pleso, SK, 2001, p. 96-97, ISBN 80-227-1542-5

[8] Dvořák L., Fučík O., Honec J., Richter M., Valenta P., Zemčík P.: *Industrial Applications of Computer Vision*, In: Automa, Vol. 2002, No. 5, CZ, p. 28-30, ISSN 1210-9592

[9] Herout A., Zemčík P., Beran V., Kadlec J.: *Image and Video Processing Software Framework for Fast Application Development*, In: Joint AMI/PASCAL/IM2/M4 workshop, Martigny, CH, 2004, p. 1

FPGA IMPLEMENTATION OF RECONFIGURABLE LICENSE PLATE DETECTION METHOD

Luděk Bryan, Otto Fučík

FIT BUT Brno,
Božetěchova 2, 612 66 Brno
Czech Republic
email: crha@fit.vutbr.cz

ABSTRACT

In this article, we propose the new object detection method with ability of self-adaptability to the environment changes using partial dynamic reconfiguration. Compared to our current method, the proposed method shows significant speed up and hit rate improvement. All parts of the proposed design have been implemented in the Virtex II XC2V500 FPGA separately. Currently we are redesigning the board to replace the Virtex FPGA with an FPGA containing more logic, which will allow us to implement the whole method together.

1. INTRODUCTION

In [1], the traffic monitoring system Unicam has been presented. The Unicam system is based on computer vision techniques; it's primary inputs are video cameras monitoring a road. It is capable of a wide variety automated traffic related tasks, such as average speed measuring, red-light violation detection and surveillance. The system is in use at many places in the Czech Republic, and new sites are being built now.

In this article, we will deal only with the low-level part of the system, an embedded system directly connected to a video-camera. The major goal of this embedded system is to detect whether or not a license plate (and consequently a car) is present in an image, and return the coordinates of the license plate.

Processing at the embedded system is provided by a DSP processor and an FPGA chip. So far, we used the FPGA chip only for simple tasks like buffering or brightness measuring; actual license plate detection is done in the DSP. Now, we plan to move most of the license plate detection burden to the FPGA, using our new detection method.

This work was partially supported by the Research plan No. MSM 0021630528 Security-Oriented Research in Information Technology. Also, thanks to Camea ltd. for supporting this research by providing us with the hardware boards and sample images.

2. THE LICENSE PLATE DETECTION METHOD

There are many publications dealing with the license plate detection. However, we didn't find any method designed for hardware implementation.

The basic idea of this method is to implement a large number of search units, each searching for one fragment of the license plate. Optimal size of the template is crucial for the proper function of the method. Small templates (3×3) are very sensitive to noise outside a license plate, thus unacceptable. Too large templates, close to the size of letters in a license plate, are very sensitive to noise inside a license plate - shapes of letters would have to be exactly the same as in the template. Experiments have shown that the best sizes of templates are ranging betweeen 5×5 and 8×8.

2.1. Preprocessing

Preprocessing should make feature extraction easy and cheap in hardware. The preprocessing unit uses a local thresholding technique. The whole operation is implemented as a set of two filters. First, edges are detected by a 3×3 non-linear filter. Output from this filter is the difference between the minimum and maximum of the pixels in the neighborhood. The second stage is actual segmentation. A non-linear 3×3 filter computes an average value of the neighborhood pixels, and outputs 1 if the pixel value is greater than the average, or 0 otherwise. An Example of a preprocessed image is in Fig. 1.

2.2. Feature Extraction

Feature extraction is the core of this method. The output is an image of the same size containing information about matching components. This output image is called the *template occurrence image*.

To implement the feature extraction unit, first we need a data structure to store information about the components we are searching for. We will store them as small binary

Fig. 1. Original image and preprocessing

Fig. 2. Feature extraction and classification

images called *templates*. The set of templates belonging to one object class will be called the *template bank*.

Then we need to create a filter that can detect a template. Basically, we need to compare similarities between two images (a template and a slice of an image). According to experiments, the most suitable operation for hardware implementation seems to be a *hit-or-miss* filter, that outputs 1 whenever image slice pixels and template pixels match. The feature extraction filter will then be a set of filters, each searching for one template from the filter bank. An example of an image after feature extraction is in Fig. 2.

2.3. Classification

Classification should be easy because it will profit from high-quality feature extraction and it will be implemented in software. The simpliest option is to search for a chunk of detected templates of license plate size. License plate position will then be determined as an area with the most templates. An example of such detection is in Fig. 2.

3. HARDWARE SCHEME

Fig. 3 shows the overall scheme. Connecting line descriptors show the format of the data. Input to the system is a serial stream of data with bit length *D* representing pixels coming from the sensor. The whole scheme works on serial basis, i.e. whenever a new pixel enters the system, a new pixel is produced at the output (with a certain delay, caused by the serial to parallel unit).

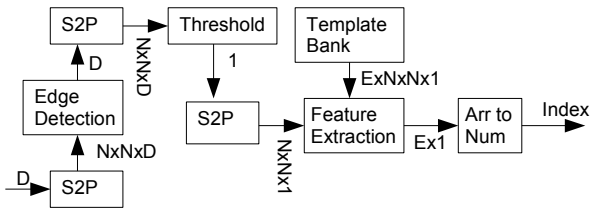


Fig. 3. Hardware scheme

Block *Serial to Parallel* (S2P) is necessary, because filters work with the neighborhood of the actual pixel. It converts serial pixel input to a matrix of $N \times N$ pixels, that contains actual pixel (delayed from input) with its surroundings. With this matrix of pixels ($N \times N$ pixels with bit depth D), we can do the preprocessing in the *edge detection unit* and then the *threshold unit*.

After preprocessing, feature extraction can take place using a template bank. It compares all E templates with an image slice, and if some of the templates fit, the proper output signal is set. There is one output signals for each template, i.e. there is E output signals from the feature extraction unit.

This output consisting of E bit vector could be already an ouput forming the template occurence image. However, memory requirements of 300 bits per pixel in average would be unacceptable. Considering that every template in the bank is unique, there will be at most 1 active bit in the vector. $ArrtoNum$ converts the E bit vector into a position of the active bit, or 0 if there is no bit active. This reduces the vector size to the maximum of $log_2(E + 1)$ bits, which is reasonable.

We have been using the DX64 boards for the developement of the proposed design. They consist of a DSP processor TMS 320C6416 and an FPGA Virtex II XC2V500. However, the FPGA will not be large enough for the whole design, so we want to redesign the board and replace the Virtex FPGA with an FPGA containing more logic. A suitable one seems to be the Spartan III XC3S1600E with a sufficient number of logic gates and a very good size/price ratio. It's also suitable for partial dynamic reconfiguration as described in section 4. Limitation is necessity of reconfiguring whole column at once, but in our case this can be easily resolved by placing all logic for reconfiguration to the whole column.

4. ADAPTABILITY TO AN ENVIRONMENT BY RECONFIGURABLE FPGA

The method proposal invokes two questions: how to create the template bank? How to update the bank if an environment (or object features) changes, and the bank doesn't give satisfactory results? There are two solutions to this question, either *static reconfiguration* or *runtime reconfiguration*. Be-

fore we discuss these two options, let's look at how we create the templates.

4.1. Creating Templates

We developed an algorithm to create a template bank from an image. The input to the algorithm is an image and the coordinates of the license plate in that image. The algorithm then creates many templates based on the license plate image. These templates are then tested to see how much they help finding the license plate - rating of a template is based on how many occurences were found in the license plate and how many outside. Templates with the highest rating are then saved to the bank. The algorithm can be applied to more images, while keeping the best templates of all images in the bank.

4.2. Static Reconfiguration

There will be predefined banks (created on a PC from real images) for different weather conditions at the location. For example, there will be only one bank for a camera in a tunnel, but there will be day/night/strong sun banks for the outside environments. The FPGA chip will then be statically reconfigured with the bank that fits best with the actual weather conditions. Static reconfigurable solution is easy and doesn't require any extra developement or tools.

4.3. Runtime Reconfiguration

Runtime reconfiguration block scheme is in Fig. 4. It is modified hardware scheme from Figure 3. Together with the normal feature extraction process (using "current" template bank) there is a parallel branch for feature extraction of new templates being tested ("test bank"). Results from both of these branches are evaluated and compared in the processor. The testing branch creates new templates for the test bank, and removes the worst templates. Normal branch adds good templates to the current bank from the test bank.

In order to replace old templates with new ones in the banks, the templates have to be in fixed predefined positions. Probably easiest is to place the templates in a regular array. Here we suggest the solution using standard Xilinx ISE tools.

4.3.1. Fitting the Templates into LUTs

Look up tables (LUTs) are basic building blocks in the Xilinx FPGAs. Every LUT is configured by 16bit array. Output of the LUT is the $n-th$ value of this array, where n is a binary number formed by the 4 LUT inputs. The template values will be stored in the LUTs arrays - all bits are reset to zeros, except of the one for which the input combination match the corresponding part of the template.

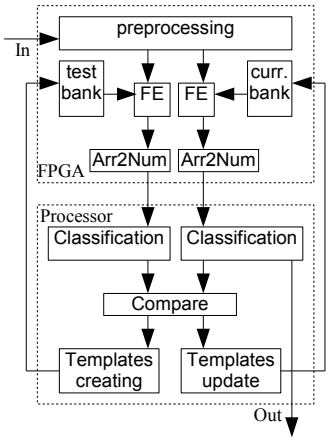


Fig. 4. Runtime reconfiguration

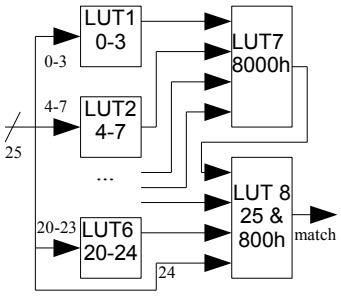


Fig. 5. Fiting a 5x5 template comparison to 8 FPGA slices

For a 5×5 template, there are 25 one-bit comparisons needed. Output is 1 when all template bits match to the input, or 0 otherwise. This function can be performed using 8 LUTs. Organization of such scheme is in Figure 5. First six LUTs are comparing 24 input bits to the template values. 7th LUT keeps the $8000h$ value which forms 4 input AND. 8th LUT works similar way as 7th, plus treating 25th bit of input.

4.3.2. Creating the Regular Slice Array of Templates

Each slice of Spartan-3 FPGA consists of two LUTs, so we can place one template into 4 LUTs. This can be achieved by setting the constraints in the UCF file. Example of implementation of one half of the template is following:

```
INST "LUT4_10" LOC=SLICE_X70Y0;
INST "LUT4_20" LOC=SLICE_X70Y0;
INST "LUT4_30" LOC=SLICE_X70Y1;
INST "LUT4_40" LOC=SLICE_X70Y1;
```

$LUT4\_n$ is the label of LUT component instatiation in the VHDL code, where n is a number of each of the 8 LUTs of the template. Last number (in this case, 0) is the tem-

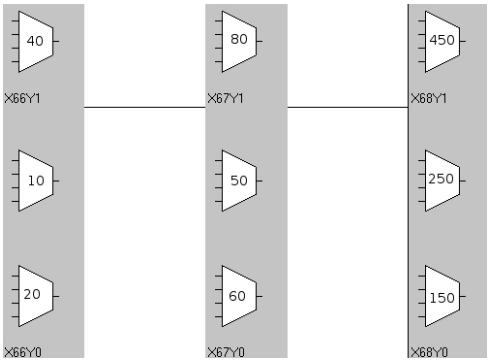


Fig. 6. Position of LUTs in slices

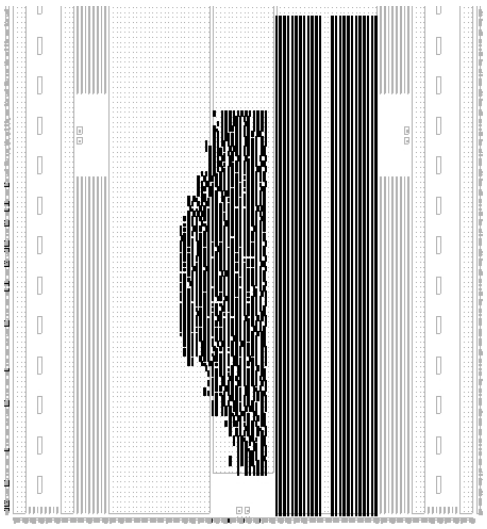


Fig. 7. Placement of two template banks and related logic

plate number. We implemented a simple C program automatically creating such a UCF file. Placement of the LUTs to the slices is shown in Figure 6. Numbers $XnYn$ show the number of a slice, numbers inside LUTs are numbers of LUTs discussed earlier in this paragraph. In the figure, you see the beginning of the first (template 0) and partly the second (template 50) column.

Placement of the design in the XC3S1600E FPGA with two banks of filters of 300 templates is shown in Figure 7.

4.3.3. Reconfiguration of Banks

As can be seen in Figure 4, we need to be able to reconfigure both current and test banks from the processor. As we have shown before, template values are stored in the LUT configuration. Thus, we don't need to reconfigure anything else but the LUTs configurations in certain columns. The most efficient way is to directly change the configuration data for the partial reconfiguration in the processor. To find out what bits in the bitstream are related to each LUT configuration, we used reverse engineering. However, this technique hasn't been tested on real hardware yet, as we have to wait for the redesign of our board.

5. EXPERIMENTAL RESULTS

We tested the method on a set of images from real traffic. Our results are compared to the method that we use currently for detection in Unicam cameras, and has been developed for many years. Testing of the method have been performed on a total of 1283 images with various weather conditions. The proposed method's accuracy was 98.6% while our current method reached 96.6%. Speed up compared to our current method running on the embedded processor TMS 320C6416 is 15.8. All accuracy and speed up results are presented and thoroughly discussed in one of the other articles (if accepted, citation will be included in the final paper version).

Proposed method shows very promising results. Although we are at the beginning of its developement and implementation, it shows better outcomes than the current method.

6. HARDWARE REQUIREMENTS

We implemented all parts of the design in the Virtex E or Virtex II FPGA, and we synthesized them for the target Spartan III XC3S1600E FPGA to estimate the total occupation of the chip. The design occupies 31% of the XC3S1600E FPGA chip, which leaves enough space for more templates and other functions. Maximum propagation delay of 24 ns allows "on-the-fly" implementation, as Unicam cameras' pixel clock is usually 40 ns.

7. CONCLUSIONS

Experimental results are very promising. The proposed method's hit rate is 98.6%, while our current method only reached the hit rate 96.6%. Main advantage of the proposed method is ability of self-adaptability to the environment changes using partial dynamic reconfiguration. Future work involves mainly implementation of the whole method to the hardware. We also plan to involve the method in another project, that deals with a visual inspection of different products on a conveyor.

8. REFERENCES

[1] Fučík O., Zemčík P., Tupec P., Crha L., Herout A.: The Networked Photo-Enforcement and Traffic Monitoring System Unicam, In: Proceedings of Engineering of Computer-Based Systems, Los Alamitos, US, IEEE CS, 2004, p. 423-428, ISBN 0-7695-2125-8

ENVIRONMENT FOR HW/SW CODESIGN OF EMBEDDED SYSTEMS

Otto Fučík, Jiří Šustek, Luděk Crha
FIT BUT Brno
Božetěchova 2, 612 66 Brno
fucik@fit.vutbr.cz, xsuste04@fit.vutbr.cz, crha@fit.vutbr.cz

Abstract. *The paper describes methodology and implementation of a HW/SW environment dedicated for codesign of embedded systems using FPGAs and CPUs in DSP applications. It integrates all necessary codesign steps of the particular HW/SW subsystems as well as of the overall system in one environment. Presented is integration and interfacing of the environment parts including VHDL simulator and C compiler. An example of using the proposed approach for real-time image processing system design is also shown.*

1 Introduction

Embedded HW/SW systems design and implementation ([1], [2], [3], and [4]) poses a significant design problem of an efficient simulation of the entire system consisting of hardware parts (e.g. written in VHDL), software (e.g. the C code), and the interface. While creating complex applications based on interacting HW and SW components, the authors were faced how to effectively co-simulate the entire system. For such purposes a simple yet powerful enough HW/SW codesign environment has been developed and is being used for creating HW/SW embedded systems for real-time image processing applications. Another very important aspect of the motivation has been to create an environment which is using tools - a VHDL simulator and C compiler and which will be given free to students for their projects in the HW/SW codesign field.

Let’s shortly describe the embedded system DX6 [2] which serves us as a basic platform for HW/SW applications implementation. There is a digital video data interface including FIFOs as well as many low-level image processing tasks, like digital filters, can be implemented in the FPGA. The FPGA is also providing Ethernet interface for easy and fast link with the host computer. The FPGA is mapped into the DSP processor memory map to create the HW/SW interface. So the software parts (running on the DSP) can communicate with hardware parts (implemented in the FPGA) very fast. The DSP processor has a sophisticated DMA controller with FIFO buffering which allows SW parts to communicate with HW parts very efficiently. All data transfers in the system are realized using the DMA support with a little or no CPU assistance. So, the CPU is fully available for real-time image processing.

The basic idea of the proposed environment is to make a standard interface between hardware parts (specified and simulated in VHDL) and software parts (coded in C) and let

them behave during simulation the same way like they finally be working in the real system. Thus the designer can efficiently simulate and test all parts of the design together. Note: the system partitioning on HW and SW parts is currently done manually by the user.

2 Implementation

Although there are many possibilities how to connect VHDL simulator and the C language compiler (e.g. FLI), there is not available a free environment that is simple and robust enough. That's why we have decided to make our own interface that fulfills our requirements.

The basic objectives of the proposed environment include compatibility with the Linux and Windows operating systems and possibility of using the Modelsim XE Starter simulator which is free for students. For the software parts implementation any C complier available can be used.

The major problem was to interface the VHDL simulator directly to the C compiler from the VHDL code. Analysis have shown that the most suitable subsidiary language which can satisfy all our needs is the TCL scripting language. TCL provides functions allowing to stop and then start again the VHDL simulator as well as to read or set the signals. As the interface medium ASCII text files are used. Advantage of this approach is its OS platform independence.

2.1 Interface model

The interface model is shown in the Figure 1. The purpose of the model is to make signals from the VHDL program accessible to C programs.

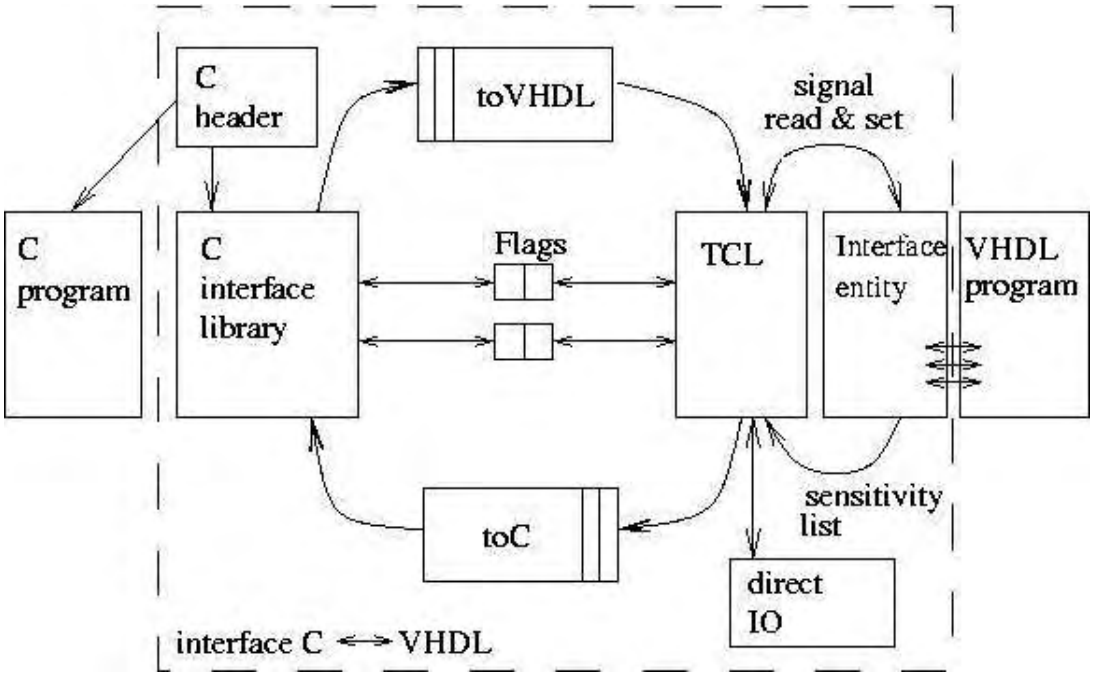


Figure 1: Interface model

The entire path of the signals consists of the following steps: The interface model must be connected to the simulated VHDL code. For this purpose, the VHDL entity called

interface entity has been created. It is connected commonly to the VHDL code by the *port map* statement. At the other side of the interface entity, there is the connection to the *TCL script*. The TCL script acts as the connection to/from the *C interface library* through the files (to C and to VHDL). To avoid concurrent read/write of a single file, *flags* are set/reset accordingly. At the C side, the C interface library provides classes and methods that allow accessing the tested *C program* to the signals of the simulated VHDL code.

2.2 Interface entity and TCL script

To transfer signals from the VHDL code further, the TCL function performing this transfer must be activated. This can be done by inserting a breakpoint to the VHDL code in the interface entity. By doing this, the running simulation is stopped upon reaching the breakpoint and the TCL function is called. This TCL function provides reading/update of the simulated signals to/from the file.

For inserting breakpoints and creating the TCL script, special macros, put to the VHDL code, are used. Subsidiary program called *compiler* parses the VHDL code of the interface entity, searches for these macros, and generates the TCL script realizing the signal transfer.

2.3 Reading signals

To illustrate how the signals are passed, the following example of reading the *data* signal by the C program when the *write* signal is set by the simulated VHDL code is shown.

```
transfer_1 : process (write)
begin
  a <= b; -- *** put(data)
end process transfer_1;
```

Example 1: Reading data signal by the C code

The assignment “a <= b” does not carry any function, it’s there just for the breakpoint to grip at the line. The C compiler searches for the “***” occurrences, and puts the breakpoints on the corresponding lines. The compiler than generates an adequate TCL function that transfers the signal – in this case *data*, and binds this function to the proper breakpoint. The advantage of this approach is the automation of the process, and also the encapsulation of the TCL code – designer doesn’t need to be familiar with this scripting language.

2.4 Writing signal values

Writing the signal values from the C code is done in a similar way as reading:

```
actualize : process
begin
  a <= b; -- *** actualize()
wait for 40 ns;
end process actualize;
```

Example 2: Writing signals from the C code

The macro named “actualize” is used for this objective. Important difference is that the function is not called right after the signal has been changed. The signal state is sampled periodically – in this case 40 ns, and forwards the signal state to the simulator.

2.5 Interface at C code side

At the C side of the interface, the library providing the classes and methods for the signal communication is linked to the C code. A signal is an encapsulated object that can be set or read through the predefined interface – methods.

In the Example 3 a simple program is shown. The 16 bits wide signal vector *data* is defined together with the 1 bit signal *irq*.

```
signal data(16); // 16 bits
signal irq(1);
x = data.getInt()
data.setHex("4F");
irq.setBin("1");
```

Example 3: The C code interface example

2.6 Data input/output stream extension

In some cases it’s necessary to work with a large amount of data. Therefore the interface for these data transfers was created, both for reading and writing. Again, the process is created in the interface entity with the macro for the file read/write, as shown in the Example 4.

```
transfer: process
begin
a <= b; -- *** getFromFile(data, file1)
wait for 40 ns;
end process transfer;
```

Example 4: File stream reading

In the Example 4, data are read from the file1 and put to the *data* signal every 40 ms.

3 Example

The proposed environment is being used for simulation and synthesis of the real-time vehicle’s license plate detection system (see [1], [3], and [4]). The goal is to find the location of a license plate present in the vehicle’s image captured by a digital camera in real-time.

The algorithm of searching for the license plate has two phases. In the first, feature extraction phase, the image is filtered by the set of digital filters implemented in hardware (FPGA). Each filter searches predefined patterns in the image. In the second phase the software parts running on the DSP processes the extracted futures and decides the results.

Let’s note that the HW parts (FPGA) and SW parts (DSP) are processed concurrently which increases the system throughput significantly and allows real-time processing of all images in the video stream.

In the Figure 2, a block diagram of the simulation of the image processing system is shown. Input and output data are provided in respective files. For communication with the DSP, regular interface functions are used.

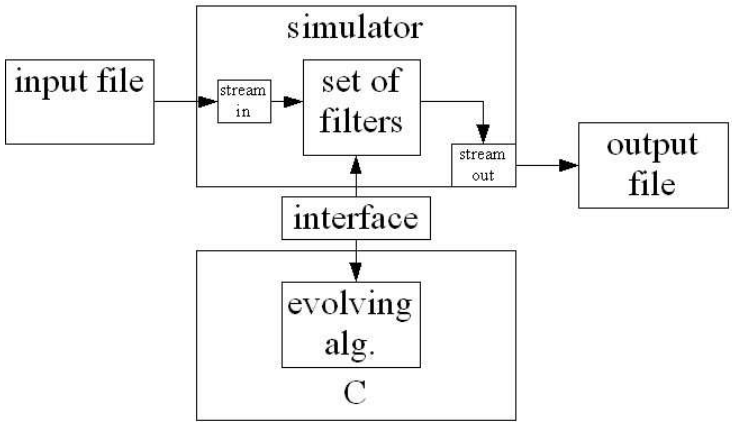


Figure 2: Simulation of the license plate detecting system

To simulate the entire system, the user is providing input data files (e.g. gathered by the DX6 system and stored on the host computer), the VHDL and C codes. Then runs the TCL script and analyzes output files with results. If the results are correct, VHDL synthesis tool and C compiler can be run and the final implementation generated and loaded into the DX6 system. Finally, using the same input data the DX6 system should generate the same results.

Conclusions

An environment dedicated for HW/SW codesign of embedded systems has been presented, which integrates all necessary subsystems in a simple environment including interfacing of VHDL simulator and C compiler. An example of a vehicle's license plate detection system development, using the proposed environment, has shown that the designer's productivity has greatly been improved and the design process simplified.

References

[1] Crha L.: System for the license plate detection and image compression using hardware , In: Proc. of the 7th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems, Bratislava, SK, SAV, 2004, pp. 3, ISBN 80-969117-9-1.

[2] Crha L., Fučík O., Zemčík P., Drábek V., Tupec P.: Inter chip communicating system with dynamically reconfigurable hardware support In: Proceedings of the 6th IEEE International Workshop on DDECS 2003, pp. 311-312, Poznan, Poland, ISBN 83-7143-557-6.

[3] Crha L., Fučík O., Drábek V.: Image filter implementation in FPGA used for the license plate, In: Proceedings of 38th International Conference MOSIS'04, Ostrava, CZ, MARQ, 2004, pp. 6, ISBN 80-85988-98-4.

[4] Fučík O., Zemčík P., Tupec P., Crha L., Herout A.: The Networked Photo-Enforcement and Traffic Monitoring System, In: Proceedings of Engineering of Computer-Based Systems, Los Alamitos, US, IEEE CS, 2004, pp. 423-428, ISBN 0-7695-2125-8.