VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY

DEPARTMENT OF MICROELECTRONICS

VLIV JAZYKŮ VYSOKÉ ÚROVNĚ NA VÝSLEDNÝ FYZICKÝ NÁVRH ČÍSLICOVÝCH OBVODŮ DO FPGA

THE IMPACT OF HIGH-LEVEL-SYNTHESIS LANGUAGES ON THE FPGA PHYSICAL DESIGNS OF DIGITAL CIRCUITS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Martin Sikora**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Martin Šťáva, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Mikroelektronika a technologie**
Ústav mikroelektroniky

Student: Martin Sikora ***ID:*** 186182
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Vliv jazyků vysoké úrovně na výsledný fyzický návrh číslicových obvodů do FPGA

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je zjistit, jaký dopad na výslednou implementaci konkrétních obvodů do čipů FPGA má použití jazyků pro vysokoúrovňovou syntézu číslicových obvodů a systémů v porovnání s jazyky pro nízkoúrovňovou syntézu. Mezi jazyky pro vysokoúrovňovou syntézu (tzv. high-level synthesis; HLS) patří např. C/C++, SystemC, Handel-C apod.; mezi jazyky pro nízkoúrovňovou syntézu patří např. VHDL, Verilog, ABEL aj.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 31.5.2018

Vedoucí práce: Ing. Martin Šťáva, Ph.D.
Konzultant:

doc. Ing. Jiří Háze, Ph.D.
předseda oborové rady

ABSTRAKT

Popularita vysokoúrovňové syntézy (HLS) se postupně zvyšuje a nástrojů pro ni stále přibývá. Otázkou je, jaký dopad mají tyto nástroje na konečný návrh číslicového obvodu a jestli se v konečném důsledku návrh v jazyce vyšší úrovně oplatí. V této práci je uveden přehled těchto nástrojů a vybrané nástroje jsou porovnávány na základě stanovených kritérií.

KLÍČOVÁ SLOVA

HLS, RTL, FPGA, VHDL, vysokoúrovňová syntéza, MachSuite

ABSTRACT

Popularity of high-level synthesis is gradually increasing and the number of tools for it is still growing. The question is, what impact do these tools have on the final digital design and whether design in high-level language will eventually pay off. This thesis presents an overview of these tools and choosen tool are then tested and compared based on the given criteria.

KEYWORDS

HLS, RTL, FPGA, VHDL, High-level synthesis, MachSuite

SIKORA, Martin. *Vliv jazyků vysoké úrovně na výsledný fyzický návrh číslicových obvodů do FPGA*. Brno, 2018, 33 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky. Vedoucí práce: Ing. Martin Šťáva, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Vliv jazyků vysoké úrovně na výsledný fyzický návrh číslicových obvodů do FPGA“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Martinu Šťávovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

V dnešní době jde vývoj technologií značně kupředu a s tím jsou spojeny i nároky na rychlost vývoje nových zařízení. V mnoha aplikacích je z důvodu minimalizace zapotřebí integrovat co nejvíce funkcí do jediného čipu. Z tohoto důvodu se v minulosti začaly vyvíjet a používat mikroprocesory a mikrokontroléry. Avšak ty nejsou optimální pro výkonnější aplikace, kde je pro rychlost zapotřebí speciálních hardwarových struktur. Ty lze realizovat speciálními obvody ASIC (Application Specific Integrated Circuit – zákaznický integrovaný obvod), které jsou ovšem časově náročné na návrh a výrobu. Více praktickými jsou obvody PLD (Programable Logic Device – programovatelné logické zařízení), jako jsou například FPGA (Field Programable Gate Array – programovatelné hradlové pole).

Z počátku se PLD obvody popisovaly jazyky nižší hradlové úrovně jako jsou ABEL, CUPL nebo PALASM. Avšak postupně se pro urychlení vývoje přešlo k jazykům s vyšší úrovní abstrakce popisu, tzv. RTL (Register-Transfer Level). Mezi nejpoužívanější tyto jazyky patří VHDL a Verilog, které se z velké části používají dodnes. Nicméně, z důvodu rychlého postupu technologie, nátlaku na time-to-market a složitosti nových obvodů, přestávají být jazyky na této úrovni abstrakce dostačující. Proto se začaly vyvíjet nástroje, které by dokázaly převést jazyky vyšší algoritmické úrovně, na úroveň RTL. Tento proces převodu se nazývá HLS (High-Level Synthesis – vysokoúrovňová syntéza).

Z počátku se experimentovalo s mnoha jazyky vyšší úrovně, ale jelikož je jazyk C/C++ velmi rozšířený, široce používaný a většina algoritmů je v něm již popsána, velká část nástrojů pro HLS se zakládá právě na něm. Dalšími HLS jazyky jsou buďto nadstavby jazyka C/C++ (SystemC, Handel-C, ImpulseC atd.) nebo nové specifické jazyky, které se vyvíjí speciálně pro určité HLS nástroje (např. BSV).

Vývoj a verifikace jsou použitím HLS značně urychleny, avšak není jasné, zda se dosahuje stejné optimalizace jako při použití jazyka nižší úrovně, jelikož optimalizaci hardwarových prostředků neprovádí designer, ale nástroj HLS. Těchto nástrojů je však v dnešní době na trhu mnoho a vybrat ten správný nemusí být jednoduché.

Cílem této práce je zjistit zda lze při návrhu v jazyce vysoké úrovně dosáhnout podobných výsledků jako u návrhu v jazyce nižší úrovně a porovnat některé z dostupných HLS nástrojů.

1 REŠERŠE

Problematikou HLS se zabývá mnoho publikací. Historii HLS například popisuje článek [8], který rozděluje vysokoúrovňovou syntézu do čtyř generací a polemizuje, proč předchozí generace neměly příliš velký úspěch a uchytila se až ta poslední.

V roce 2008 vyšla kniha [11], která popisuje postup od algoritmu k digitálnímu obvodu a také se zabývá popisem několika HLS nástrojů, avšak velká část z nich od té doby prošla velkými změnami.

Článek [9] z roku 2009 prezentuje zkušenosti návrháře s nástroji HLS. Tato práce poukazuje na to, že se HLS stále vyvíjí, a pro optimální řešení úlohy je nutné experimentovat s více nástroji.

Na téma porovnání nástrojů pro vysokoúrovňovou syntézu vyšel v roce 2012 článek [1]. Ten porovnává 12 nástrojů na základě jednoduchosti implementace vstupního kódu, úrovni abstrakce, datových typů, snadnosti učení, dokumentace, výsledné plochy a nástrojů pro verifikaci. Tyto informace jsou přehledně zobrazeny v pavučinových grafech. Tento článek dochází k závěru, že HLS usnadňuje návrh a verifikaci digitálního obvodu, avšak je ještě stále co zlepšovat a chybí standardizace vstupu, což nutí designera, školit se pro každý specifický nástroj.

Další článek [2], který vyšel v roce 2016, porovnává větší množství komerčních nástrojů s univerzitními nástroji na základě optimalizace výsledného RTL popisu, vygenerovaného ze sedmnácti různých benchmarků (testovacích vstupních souborů). Závěrem je, že komerční a univerzitní nástroje se moc neliší z pohledu kvality, ale komerční nástroje podporují více funkcí a jsou více robustní.

Porovnáváním a certifikací HLS nástrojů se také zabývá firma Berkeley Design Technology Inc.. Prozatím byly na jejich webových stránkách zveřejněny testy nástrojů AutoPilot [12] (nyní Vivado HLS) a Synphony C Compiler [13].

2 TEORETICKÉ ZPRACOVÁNÍ

Tato kapitola se zabývá teoretickým rozborem nastoleného problému. Popisují se zde jednotlivé nástroje vysokoúrovňové syntézy, metodika testování a testovací kritéria.

2.1 High-level synthesis

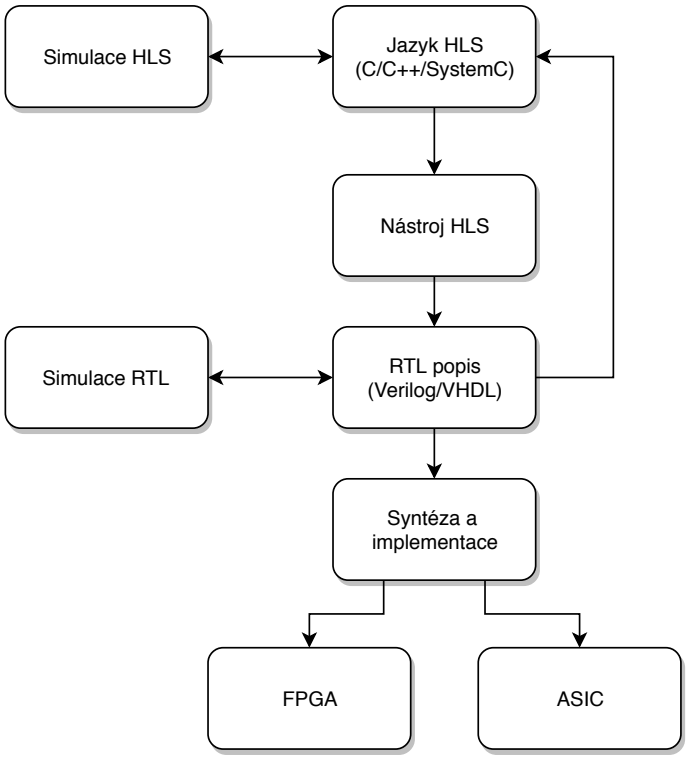
Číslicový obvod může být vytvořen na několika úrovních abstrakce. Běžně používané úrovně abstrakce jsou hradlová úroveň, RTL úroveň a algoritmická úroveň.

Vysokoúrovňová syntéza je automatizovaný designový proces, který interpretuje algoritmický popis požadovaného chování a vytvoří hardware, který implementuje toto popsané chování. Počáteční bod HLS návrhu je kód v jazyce vyšší úrovně jako je například C/C++/SystemC. Kód je analyzován a přeložen do RTL popisu, který se obvykle dále syntetizuje na hradlovou úroveň. Cílem HLS je umožnit hardwarovým designerům efektivně popsat a ověřit hardware tím, že jim dá lepší kontrolu nad optimalizací jejich navrhované architektury a dovoluje designerovi popisovat návrh na vyšší úrovni abstrakce, zatímco vysokoúrovňový nástroj implementuje RTL popis [14].

Zatímco logická syntéza používá RTL popis designu, HLS pracuje na vyšší úrovni abstrakce, začínající algoritmickým popisem v jazyce vyšší úrovně. Designer obvykle popisuje funkcionalitu modulu a vytváří propojovací protokol. Nástroje pro HLS zpracují architekturu a přetransformují nečasovaný nebo částečně časovaný funkční kód do plně časované RTL implementace. Tyto RTL implementace jsou následně použity tak, jako v tradičním postupu logické syntézy k vytvoření implementace na hradlové úrovni [14].

Vysoká úroveň abstrakce zároveň poskytuje designérovi rychlejší a snadnější simulace. Vše se musí ovšem pro ověření také simulovat na RTL úrovni. Tento postup HLS návrhu je vidět na obrázku 2.1.

Vyšší úrovní abstrakčního popisu se dosahuje menšího kódu a tím i kratšího času simulace. Například obvod obsahující milión hradel potřebuje kolem 300 tisíc řádků kódu v popisu RTL avšak pouze 40 tisíc řádků v jazyce C. Simulace takového RTL kódu může zabrat v průměru desetkrát až stokrát více času než ekvivalentní kód v behaviorálním popisu ve vyšším jazyce [16].



Obr. 2.1: Postup při návrhu HLS.

2.2 Přehled HLS nástrojů

V této podkapitole se nachází přehled nejznámějších aktuálně vyvíjených HLS nástrojů (viz tab. 2.1) a jejich popis. U většiny nástrojů není specifikovaná cena, jelikož ji vývojáři neuvádějí a poskytují pouze individuální nabídky na vyžádání.

Tab. 2.1: Přehled nástrojů HLS

Nástroj	Výrobce	Vstup	Výstup	Cena
CyberWorkBench	NEC	ANSI-C/SystemC	Verilog/VHDL	-
BlueSpec	BlueSpec	BSV	Verilog	-
CoDeveloper	Impulse	Impulse C	Verilog/VHDL	-
DK Design suite	Mentor	Handel C	Verilog/VHDL	-
eXCite	Y Exploration	ANSI-C	Verilog/VHDL/SystemC	-
ROCCC	Jackquard Computing	C	VHDL	open source
Vivado HLS	Xillinx	C/C++/SystemC	Verilog/VHDL	Lite Edition
Catapult HLS	Mentor	C/C++/SystemC	Verilog/VHDL	-
Stratus HLS	Cadence	C/C++/SystemC	RTL	-
LegUp	LegUp Computing	C	Verilog	-
Bambu	Politecnico di Milano	C	Verilog/VHDL	open source
GAUT	U. Bretagne	C/C++	VHDL	open source
Synphony	Synopsys	C/C++	Verilog/VHDL/SystemC	-
Intel HLS Compiler	Intel	C++	RTL	WebPack

2.2.1 CyberWorkBench

Firma NEC Corporation vyvíjí behaviorální syntézu založenou na jazyce C nazývanou Cyber již od 80. let. NEC zároveň vyvíjí verifikační a simulační nástroje specifické pro tento nástroj. Vše je integrováno do jediného integrovaného vývojového prostředí, které se nazývá CWB (CyberWorkBench) [16].

Vstupem do CWB je kód v jazyce ANSI-C nebo SystemC, který se syntézou převede na RTL popis v jazyce Verilog nebo VHDL. CWB podporuje dva druhy módy časování. V módu manuálního časování vymezí designer ve zdrojovém kódu hodinové cykly. V automatickém módu probíhá časování automaticky, ale designer má možnost popsat přesné chování cyklu tam kde potřebuje. To znamená, že může být specifikován mix nečasovaného a časovaného chování cyklů [1].

CWB také umožňuje vygenerovat technologické knihovny pro HLS na základě externích RTL knihoven. Z těchto technologických knihoven následně CWB generuje pro HLS design knihovny funkcí a pamětí. Designer má také možnost omezit počet využitých prostředků (násobiček, sčítaček atd.) k vyvážení výsledné plochy a odezvy obvodu [1].

2.2.2 BlueSpec Development Workstation

BlueSpec je nástroj společnosti BlueSpec Inc., který používá poněkud odlišný přístup k abstrakci. Design je tvořen ve speciálním jazyce BSV BlueSpec System Verilog, který je založen na Verilogu. Moduly se v něm implementují jako soubory pravidel. To znamená, že existující algoritmy, které jsou již popsány v jiných jazycích vysoké úrovně, nelze použít. Aby designeři získali syntetizovatelný kód, musí tyto algoritmy reimplementovat [1].

Tyto pravidla se nazývají GAA (Guarded Atomic Actions). Ty definují nezávislé akce a jsou používány jako behaviorální výrazy. Jelikož jsou na sobě nezávislé, tak BSV kód zůstává velmi čitelný. Hodinové a resetovací signály jsou v programu implicitní, takže nejsou pro designera viditelné [1].

Všechny designové elementy jsou implementovány jako samostatné moduly, což může vést k delším portmapám a propojovacím seznamům. Výměna dat mezi moduly je implementována v objektově orientovaném stylu použitím metod rozhraní. Pro konkrétní hardwarové komponenty, jako je například BlockRAM, používá Bluespec přístupové rozhraní server/klient, což může vést k poměrně složitému kódu pro základní operace [1].

K získání funkčně správného návrhu, musí designér pochopit, jak kompilátor generuje RTL popis ze vstupního souboru BSV. Tento požadavek, v kombinaci s vlastním designovým jazykem a neobvyklým paradigmem programování založeným na pravidlech, má za následek strmou křivku učení [1].

2.2.3 Impulse CoDeveloper

CoDeveloper od firmy Impulse Accelerated Technologies poskytuje pracovní postup C-to-gate, kde jsou algoritmy popsány v jazyce Impulse-C, mírně upraveném kódu C, a poté kompilovány jako software pomocí standardního C kompilátoru nebo přeloženy na RTL popis [3].

Kompilátor CoDeveloper používá model komunikačního sekvenčního procesu (CSP). Algoritmus je popsán pomocí kódu ANSI C a knihoven specifických funkcí. Komunikace mezi procesy se provádí hlavně datovými toky nebo sdílenými pamětmi. Poskytnuté rozhraní API (Application Programming Interface – rozhraní pro programování aplikací) obsahuje potřebné funkce pro vyjádření procesní paralelizace a komunikace, jelikož standardní jazyk C nativně nepodporuje souběžné programování [3].

Jakmile je algoritmus napsán, může být kompilován pomocí libovolného standardního kompilátoru C. Každý z definovaných procesů je přeložen do softwarového vlákna. Tímto způsobem lze celou aplikaci spustit a ověřit její funkčnost. Ladění a profilování algoritmu je proto jednoduché a lze provádět pomocí standardních nástrojů [3].

2.2.4 DK Design Suite

DK Design Suite je poněkud starší nástroj HLS, který byl získán v roce 2009 společností Mentor Graphics. V té době Mentor oznámil, že bude podporovat DK Design Suite pouze dokud zákazníci nepřejdou na jejich Catapult C, avšak později na svých webových stránkách oznámil pokračující podporu a významné vylepšení pro tento nástroj [1].

DK Design Suite používá jazyk Handel-C, který je založen na bohaté podmnožině jazyka C rozšířeného o hardwarově specifické jazykové konstrukty. Designer však musí specifikovat požadavky na časování a popsat paralelní a synchronizační segmenty v kódu explicitně. Navíc musí být mapování dat do různých pamětí provedeno manuálně. Z důvodu těchto jazykových dodatků potřebuje návrhář pokročilé hardwarové znalosti [2].

DK se orientuje především na FPGA obvody a ve zdrojovém kódu musí být zahrnut například výběr komponent a dokonce i mapování pinů. To způsobuje, že je těžší kód přenášet na jiné platformy [1].

DK design suite generuje popis v jazyce VHDL, Verilogu nebo jako namapovaný návrh pro FPGA. Latenci a propustnost finálního návrhu může designer poměrně dobře vyladit, i když s velkým množstvím manuálního úsilí [1].

2.2.5 eXCite

eXCite je nástroj firmy Y Explorations Inc., který přebírá kód v ANSI-C a syntetizuje jej na Verilog nebo VHDL popisy RTL, které jsou dále vhodné jako vstup do nástrojů pro logickou syntézu FPGA nebo ASIC [22].

Před použitím kódu C pro vysokoúrovňovou syntézu je nutné identifikovat a vložit kanály, které specifikují, jak bude generovaný hardwarový blok komunikovat s okolím. Z kanálů se čte nebo zapisuje pomocí jednoduchých volání procedur C, které se přidávají do kódu. Systém eXCite převezme tyto kanály a na základě cílové hardwarové platformy automaticky zjistí, jak je lze propojit se sběrnicemi cílové platformy [22].

2.2.6 ROCCC

The Riverside Optimizing Compiler for Configurable Circuits (ROCCC) je open source HLS nástroj vyvinut na Kalifornské univerzitě. Syntetizér přijímá přísnou podmnožinu jazyka C a generuje z ní VHDL popis. Jako vývojové prostředí ROCCC využívá volně stažitelný program Eclipse. Do něj se ROCCC implementuje jako zásuvný modul.

ROCCC nebyl navržen tak, aby vytvářel hardware pro celé aplikace, ale místo toho se zaměřuje na kritické oblasti velkých softwarových systémů. Kritické oblasti se obvykle sestávají z vnořených smyček, které provádí rozsáhlé výpočty na velkém množství dat [4].

Cílem ROCCC je maximalizovat propustnost, omezit přístupy k paměti a minimalizovat velikost generovaného obvodu. ROCCC to dělá tím, že kompiluje kód, který využívá silné stránky FPGA a zároveň omezuje typ kódu, který je na FPGA neúčinný. Konkrétně ROCCC využívá rozsáhlé množství paralelismů dostupných na FPGA a schopnost implementovat velké výpočetní řetězové zpracování toků dat, přičemž se snaží minimalizovat čtení externí paměti [4].

ROCCC analyzuje, která data jsou načítány z paměti, a pokud jsou stejné prvky znovu použity v následné iteraci smyčky, budou spíše uložena v obvodu, než znovu načtena z paměti. To je velmi důležité, jelikož je rychlost pamětí ve výpočetních systémech jedním z hlavních problémů. To činí ROCCC obzvláště užitečným pro algoritmy pracující s tokem dat a algoritmy s posuvným okénkem [1].

2.2.7 Vivado HLS

Vivado HLS, původně AutoPilot, byl zprvu vyvíjen společností AutoESL, dokud nebyl v roce 2011 převzat společností Xilinx. Nový zdokonalený produkt, který je založen na LLVM, byl vydán počátkem roku 2013. Obsahuje kompletní návrhové

prostředí s bohatou nabídkou funkcí pro jemné vyladění procesu generování z jazyka vyšší úrovně na RTL. Jako vstup jsou přijímány C, C++ nebo SystemC kódy a výstupem jsou hardwarové moduly, které jsou generovány v VHDL, Verilogu a SystemC [2].

Aby bylo dosaženo optimalizovaného návrhu, prochází nástroj několika fázemi. Vivado naplánuje logické operace pro každý hodinový cyklus a pro každou operaci přiřadí hardwarové prostředky. Z návrhu nástroj extrahuje řídicí logiku a vytvoří konečný stavový automat, kterým se v návrhu RTL operace seřadí [5].

Nástroj Vivado HLS syntetizuje funkce z jazyka C do bloků v RTL hierarchii. Argumenty funkce nejvyšší úrovně se podle potřeby syntetizují do vstupních a výstupních portů, doprovázených vhodnými signály, pro navázání komunikace. Nástroj HLS umožňuje vývojářům také analyzovat a optimalizovat design. Také se po syntéze vygeneruje sada hlášení, které lze dále používat k analýze implementace [5].

Během procesu syntézy je možné aplikovat různé optimalizace, jako například řetězení operací, řetězové zpracování smyček a rozvinutí smyček. Kromě toho může být do paměti specifikováno různé mapování parametrů. Pro zjednodušení integrace výsledného akcelerátoru jsou podporovány rozhraní pro tok dat nebo také sdílená paměť [2].

2.2.8 Catapult HLS

Catapult HLS je široce používaný nástroj firmy Mentor, který byl zpočátku orientován primárně na vývojáře hardwaru ASIC. Nyní se však zaměřuje také na FPGA. Nástroj nabízí flexibilitu při výběru cílové technologie, externí knihovny, nastavení frekvence designu, mapování parametrů funkcí do registrů, paměti RAM, paměti ROM nebo rozhraní pro tok dat [2].

Catapult nabízí vynikající uživatelské rozhraní, které vede designera procesem návrhu HLS. Jednotlivé kroky zahrnují výběr zdrojových souborů, nastavení designu (cílová technologie, hodinová frekvence atd.), návrhová omezení (plocha, latence, I/O, mapování polí a optimalizace smyček), časování a generace RTL. Většina možností je nastavena rozumnými výchozími hodnotami, což usnadňuje zahájení návrhu. Kromě grafického uživatelského rozhraní může být nástroj také spuštěn ze skriptu [1].

Nástroj Catapult HLS je používaný mnoha velkými firmami, jako jsou například AMD, Bosh Visiontec, Google Inc., NVIDIA Corp., Qualcomm a STMicroelectronics [19].

2.2.9 Stratus HLS

Stratus HLS, původně C-to-Silicon, je HLS nástroj od firmy Cadence. Pomocí nástroje Stratus HLS mohou inženýrské týmy rychle navrhnout a ověřit RTL implementace z abstraktních modelů v jazyce SystemC, C nebo C++. V prostředí Stratus IDE lze tyto modely snadno vytvořit, přesměrovávat na nové technologické platformy a jednodušeji znovu použít, než tradiční ručně psané RTL. Stratus IDE také umožňuje, v prostředí syntézy na vysoké úrovni, vytvářet kompromisy mezi spotřebou, plochou a výkonem [20].

2.2.10 LegUp

LegUp je akademický HLS kompilátor, vyvinut na univerzitě v Torontu, který byl poprvé vydán v roce 2011. LegUp je navržen speciálně pro různé rodiny FPGA společnosti Altera. Nástroj přijme jako vstup program jazyka C a pracuje jedním ze dvou způsobů. První možnost je, že nástroj syntetizuje celý kód C na RTL popis. Druhou možností je syntetizování kódu do hybridního systému obsahujícího procesor (MIPS nebo ARM) a jeden nebo více hardwarových urychlovačů. Komunikace mezi procesorem a urychlovači probíhá přes paměťově mapované rozhraní na čipové sběrnici společnosti Altera [2].

Pro HLS syntézu je podporována většina jazyka C, s výjimkou dynamické alokace paměti a rekurze. LegUp je postaven v rámci kompilátoru LLVM. Ve srovnání s ostatními nástroji HLS má LegUp několik jedinečných funkcí. Podporuje platformy Pthreads a OpenMP, kde jsou paralelní softwarové podprocesy automaticky syntetizovány do paralelně pracujícího hardwaru [2].

2.2.11 Bambu

Bambu je akademický open source nástroj HLS vyvinutý na univerzitě v Politecnico di Milano a byl poprvé zveřejněn v roce 2012. Bambu přijímá jako vstup kód v jazyce C, který má být implementován, a konfigurační soubor XML (eXtensible Markup Language). Jako výstup produkuje odpovídající popis v HDL, skripty pro logickou syntézu spolu s testovacími soubory a skripty pro RTL simulaci. Bambu může také generovat vhodné rozhraní a datové konstrukce pro integraci vygenerovaného urychlovače se zbytkem systému [2, 6].

Pro provedení syntézy kódu, Bambu generuje jeden modul pro každou z funkcí samostatně. Proces HLS lze snadno přizpůsobit. Uživatel může rozhodnout, který z dostupných algoritmů musí být použit pro každou fázi procesu, pořadí jejich provedení a příslušné parametry [6].

2.2.12 GAUT

GAUT je open-source HLS nástroj určený pro DSP Digital Signal Processing – digitální zpracování signálu aplikace. Vstupem do GAUT nástroje je algoritmická bitově přesná specifikace napsaná v jazyce C/C++, omezení propustnosti a perioda hodinového signálu [7].

Architektura vygenerovaných hardwarových komponent se skládá ze tří hlavních funkčních jednotek: procesní jednotka, paměťová jednotka a komunikační jednotka. Procesní jednotka je datové cesta složená z logických a aritmetických operátorů, paměťových prvků, řídicí logiky a konečného stavového automatu. Úložné prvky procesní jednotky mohou být sémantické paměti (FIFO, LIFO) nebo registry. Paměťová jednotka se skládá z paměťových bank a jejich přidružených řadičů. Komunikační jednotka obsahuje synchronizační procesor a operační paměť, což umožňuje používat globálně asynchronní lokálně synchronní komunikační rozhraní [7].

Výsledkem je několik automaticky vygenerovaných modelů RTL: VHDL, SystemC CABA (Cycle Accurate Bit Accurate) a TLM-T (Transaction Level Model with Timing) s příslušnými testovacími soubory [7].

2.2.13 Synphony C Compiler

Synphony C Compiler je HLS nástroj firmy Synopsys, založený na bývalém nástroji PICO od společnosti Synfora. [1] Synphony C Compiler přijímá širokou podmnožinu jazyka C/C++, která zahrnuje běžné datové typy s pevnou desetinnou čárkou. Uživatelé se mohou rychle vyvíjet syntetizovatelné algoritmy používáním přirozenějšího sekvenčního stylu psaní kódu, včetně použití vnořených smyček a více úrovní hierarchie [15].

Jádro nástroje Synphony C Compiler aplikuje optimalizace na architektonické úrovni pro dosažení uživatelem specifikovaných cílů rychlosti, plochy a výkonu v RTL implementaci. Nejvýznamnější optimalizace jsou single-to-multi-thread transformace, které vytvářejí paralelismus ze sekvenčního kódu C. Synphony používá jedinečné víceúrovňové kompilační techniky, které umožňují tyto paralelizace. Ty mohou vznikat ve víceúrovňových vnořených smyčkách, včetně smyček, které se mohou šířit více úrovněmi hierarchie v C/C++ zdroji [15].

2.2.14 Intel HLS Compiler

Kompilátor Intel HLS Compiler je nástroj pro syntézu na vysoké úrovni, který přijímá vstup v jazyce C++ a generuje kód RTL, který je optimalizován primárně pro Intel FPGA. Tento nástroj urychluje čas verifikace oproti RTL, zvyšováním úrovně

abstrakce pro návrh hardwaru FPGA. Modely vyvinuté v jazyce C++ jsou typicky řádově rychlejší než RTL [18].

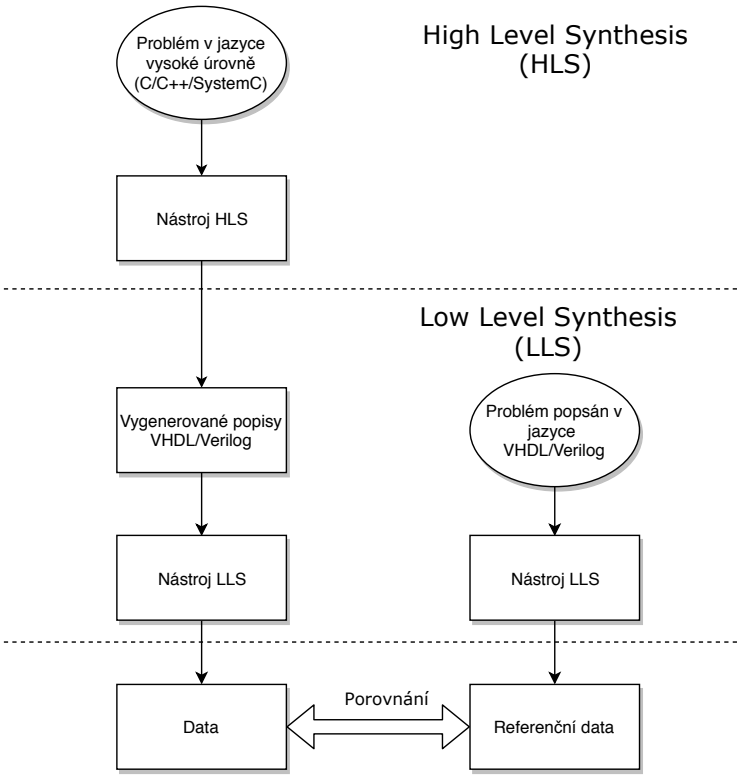
Avšak vstupní kód má určitá omezení. Kompilátor nemůže syntetizovat kód pro alokaci dynamické paměti, virtuální funkce, funkční ukazatele a funkce knihoven C++ nebo C s výjimkou podporovaných matematických funkcí [17].

2.3 Testovací metodika

Pro testování byly navrženy dvě metody z nichž jedna se zaměřuje na zjištění vlivu HLS na výsledný návrh HLS, zatímco druhá je spíše zaměřena na porovnávání HLS nástrojů mezi sebou.

2.3.1 Metoda pro zjištění vlivu HLS

Tato metoda spočívá v převodu daných problémů (algoritmů) popsaných v jazyce vyšší úrovně vybraným HLS nástrojem. Výsledný RTL popis je poté dále převeden nástrojem LLS. Zároveň je také paralelně převáděn stejný algoritmus popsaný v jazyce nižší úrovně. Po tomto převodu jsou získány data, která jsou následně porovnávána. Celá metoda je zobrazena na obr. 2.2.



Obr. 2.2: Metoda porovnání HLS a LLS.

Pro testování touto metodou byly vybrány algoritmy, které byly nalezeny jak v popisu v jazyce vyšší úrovně (v tomto případě SystemC), tak i v popisu v jazyce v nižší úrovně (VHDL/Verilog). Seznam těchto problému a jejich popis je uveden v tabulce. 2.2.

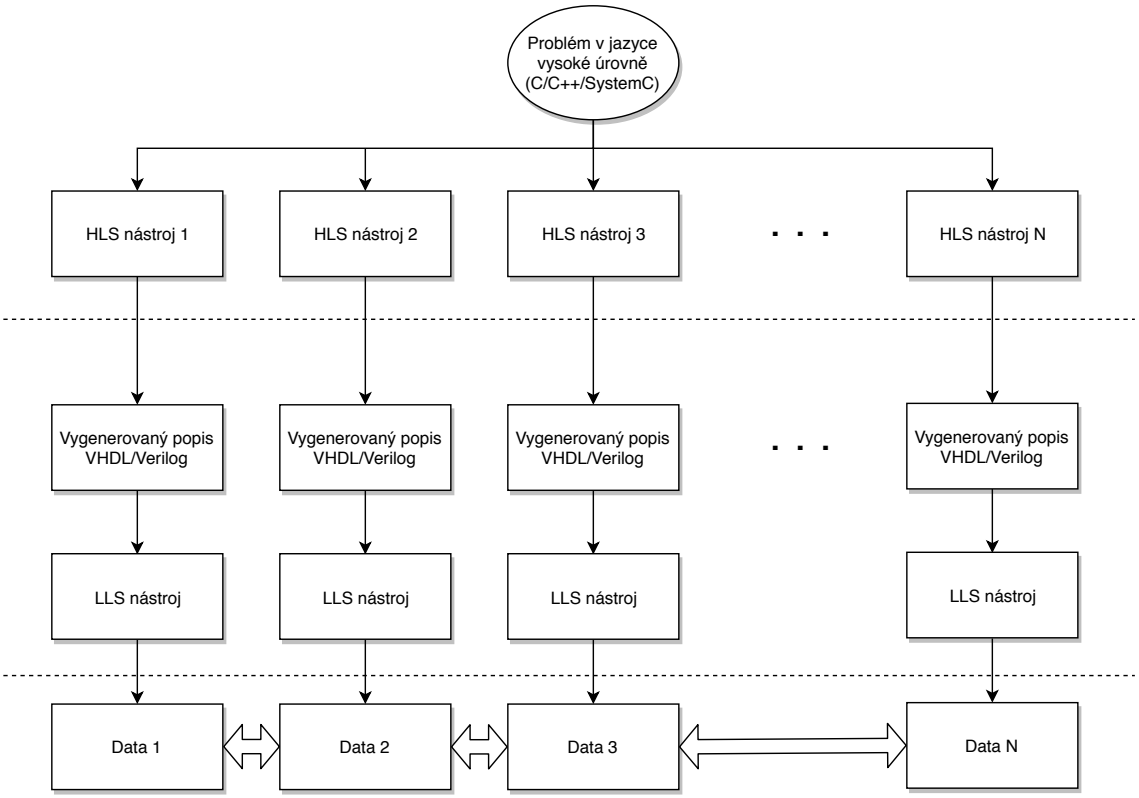
Tab. 2.2: Přehled testovaných problémů.

Algoritmus	Popis
adpcm	Adaptivní rozdílová pulzně kódová modulace.
aes	Standard pokročilého šifrování.
md5	Hašování dat.
uart	Univerzální asynchronní rozhraní.
viterbi	Algoritmus dynamického programování.

2.3.2 Metoda porovnání HLS nástrojů

Tato metoda je navržena pro porovnání efektivity jednotlivých HLS nástrojů mezi sebou. Problémy popsané v jazyce vyšší úrovně budou převedeny nejprve na popis RTL a poté bude jednotným nástrojem provedena nízkoúrovňová syntéza. Poté se budou porovnávat výsledná data mezi sebou. Metoda je zobrazena na obr. 2.3

Pro toto testování budou použity problémy popsané v jazyce C, jelikož jej podporuje většina HLS nástrojů. Konkrétně bude použit testovací balíček MachSuite [23], který je vyvíjen na Harvardské univerzitě. MachSuite je sada 19 různých algoritmů psaných v jazyce C zahrnujících 12 různých jader (viz tab. 2.3), psaných pro pokrytí různorodých domén aplikací a pro začlenění různých algoritmických možností.



Obr. 2.3: Metoda porovnání HLS nástrojů.

Tab. 2.3: Přehled benchmarků MachSuite

Jádro	Algoritmus	Popis
AES	AES	AES šifrování
BACKPROP	BACKPROP	Učení neuronové sítě
BFS	BULK	Prohledávání do šířky
BFS	QUEUE	Prohledávání do šířky
FFT	STRIDED	Rychlá Fourierova transformace
FFT	TRANSPOSE	Rychlá Fourierova transformace
GEMM	NCUBED	Násobení matic
GEMM	BLOCKED	Násobení matic
KMP	KMP	Porovnávání řetězců
MD	KNN	Molekulární dynamika
MD	GRID	Molekulární dynamika
NW	NW	Zarovnání DNA
SORT	MERGE	Řazení
SORT	RADIX	Řazení
SPMV	CRS	Řídké matice/Násobení vektorů
SPMV	ELLPACK	Řídké matice/Násobení vektorů
STENCIL	STENCIL2D	Výpočet šablony
STENCIL	STENCIL3D	Výpočet šablony
VITERBI	VITERBI	Algoritmus dynamického programování.

2.4 Vyhodnocovací kritéria

Vyhodnocení bude stanoveno na základě kritérií, které jsou stanoveny a popsány v této kapitole. Mezi tyto kritéria patří množství využitých prostředků FPGA, frekvence a případně spotřeba.

Hlavními prostředky FPGA obvodů jsou LUT (LookUp Table – náhledová tabulka), FF (Flip-Flop – Klopný obvod), bloky paměti RAM a DSP bloky. Náhledové tabulky jsou základními stavebními bloky FPGA. Fungují na principu paměti, která má pro každou kombinaci vstupů definovaný výstup. Lze pomocí nich vytvořit libovolnou logickou funkci, která je ovšem omezená počtem vstupů (běžně 4 nebo 6). LUT lze také v FPGA využít jako distribuovanou paměť, která má výhodu asynchronního čtení. Klopné obvody, nebo také registry, jsou druhým základním prostředkem FPGA. Konkrétně se jedná o klopné obvody typu D, které se využívají k vytvoření synchronního designu. Blokové paměti RAM jsou bloky, které slouží výhradně k uchování a čtení dat. Bloky DSP jsou pomocné bloky, které obsahují hardwarovou implementaci matematických operací jako jsou sčítačky a násobičky.

Dalším kritériem je maximální dosažitelná frekvence. Tento parametr závisí na optimalizaci nejdelší kombinační cesty mezi klopnými obvody. Signál musí dorazit k následujícímu klopnému obvodu dříve než náběžná hrana hodinového signálu a proto musí být perioda clocku vždy delší. V případě nesplnění této podmínky může dojít k nestabilitě obvodu.

Spotřeba je v dnešní době miniaturizace důležitým parametrem, jelikož jsou zařízení čím dál tím více napájena z baterie. Spotřeba souvisí přímo s počtem využitých prostředků FPGA, pracovní frekvenci a cílové platformě. Tento parametr je proto syntetizačním nástrojem pouze odhadován a může se proto od skutečné spotřeby dosti lišit.

3 PRAKTICKÁ ČÁST

Tato kapitola se zabývá praktickým zpracováním nastoleného problému. Jsou zde uvedeny výsledky provedených syntéz a vybraná data jsou pro přehlednost zobrazena v grafech.

3.1 Vliv HLS na výsledný návrh FPGA

Vliv HLS na výsledný návrh FPGA se testoval dle metody uvedené v sekci 2.3.1. Zdrojové kódy v jazyce vyšší úrovně zde byly převedeny nástrojem Vivado HLS do RTL popisu. Pro syntézu z jazyka RTL byl poté použit nástroj stejné firmy - Vivado. Jako cílová platforma byl zvolen obvod XC7A200T, který spadá do rodiny Artix-7 a cílová frekvence byla stanovena na 200 MHz.

V tabulce 3.1 jsou uvedeny výsledky dosažené HLS syntézou z jazyka vyšší úrovně (SystemC) a v tabulce 3.2 jsou uvedeny výsledky při použití jazyka na úrovni RTL(Verilog).

Tab. 3.1: Využité prostředky FPGA při syntéze z jazyka SystemC.

Jádro	LUT	FF	BRAM	DSP	Frekvence [MHz]	Spotřeba [W]
adcpm	112	72	1	0	207,8	0,137
aes	3046	3155	2	0	202,8	0,238
md5	341	288	0	0	264,2	6,118
uart	206	142	0	0	196,5	1,642
viterbi	811	1039	32	0	310,4	1,2

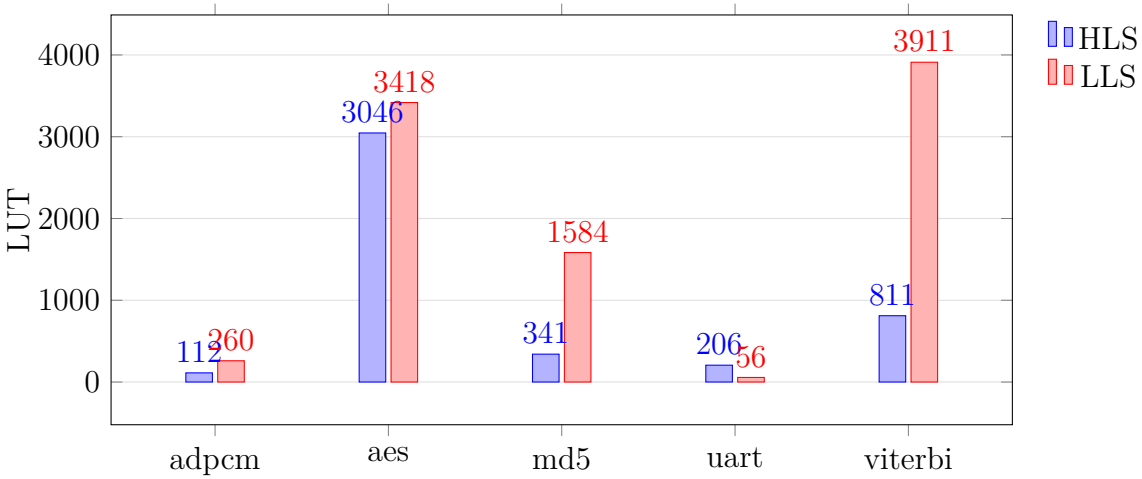
Tab. 3.2: Využité prostředky FPGA při syntéze z jazyka Verilog.

Jádro	LUT	FF	BRAM	DSP	Frekvence [MHz]	Spotřeba [W]
adcpm	260	85	1	0	162,0	0,15
aes	3418	2992	0	0	124,6	0,186
md5	1584	910	0	0	77,8	0,179
uart	56	55	0	0	282,3	0,134
viterbi	3911	1512	0	0	174,8	0,214

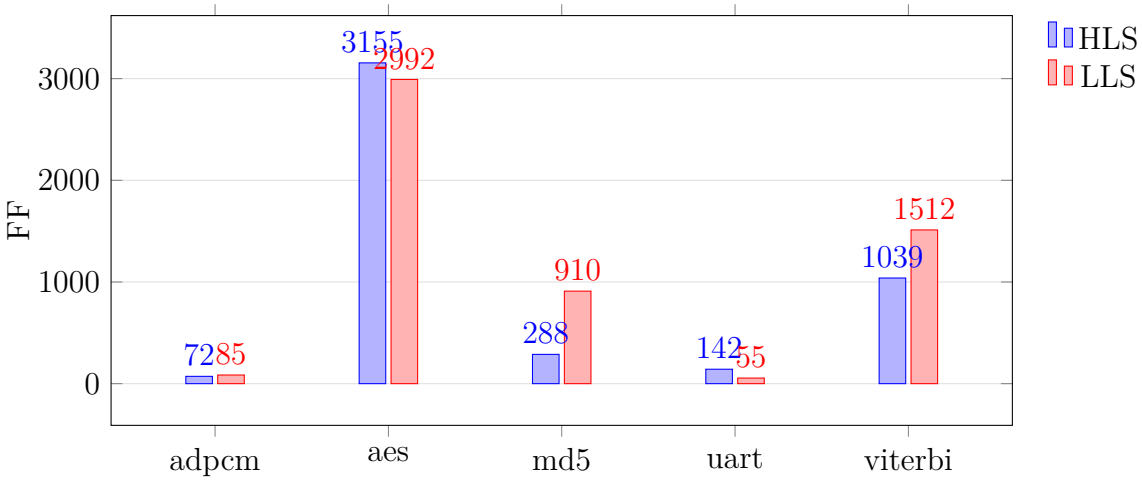
Pro lepší přehlednost byly počty využitých LUT, FF a maximální frekvence vyneseny do grafů (viz obr. 3.1 až obr. 3.3). Až na jeden případ (uart), byl návrh číslicového obvodu v jazyce vyšší úrovně lépe optimalizovaný a zabíral méně hardwarových prostředků. Co se týče maximální dosažené frekvence, návrh na vysoké úrovni opět předčil návrh v RTL, téměř ve všech případech. V obou případech nebyly využity žádné bloky DSP. Největší rozdíl návrhů nastal u algoritmů md5 a

viterbi, kde se z návrhu na úrovni RTL vytvořila spíše řetězová struktura, čímž došlo k velkému nárůstu využitých prostředků a zároveň i ke snížení frekvence.

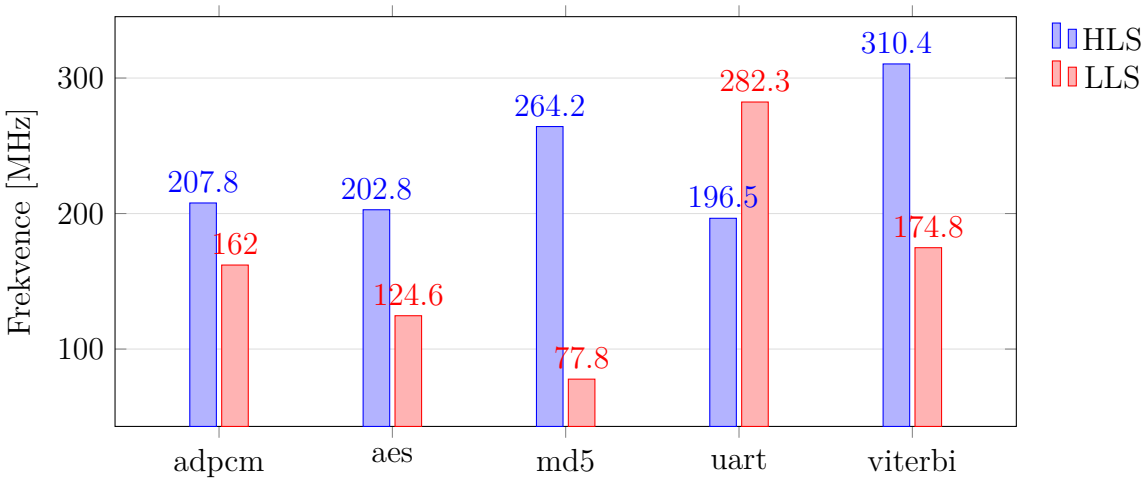
Na druhou stranu návrh v jazyce nižší úrovně měl ve všech případech menší odhadovanou spotřebu, což může být pro některé aplikace důležité. Avšak jedná se pouze o odhad po implementaci a skutečná spotřeba bude pravděpodobně vyšší.



Obr. 3.1: Porovnání využití buněk LUT.



Obr. 3.2: Porovnání využití buněk FF.



Obr. 3.3: Porovnání využití buněk FF.

3.2 Porovnání HLS nástrojů

Tato kapitola obsahuje výsledky testování podle metody popsané v sekci 2.3.2. Z dostupných nástrojů se podařilo provést syntézu pouze na nástrojích Vivado HLS, Bambu, GAUT, Intel HLS. Ostatní dostupné nástroje nedokázaly zpracovat připravené soubory v jazyce C, nebo vyžadovaly speciální syntaxi, což by znamenalo kompletní přepracování všech kódů. Jako nástroj pro LLS zde byl použit nástroj ISE Design Suite firmy Xilinx a jako cílová platforma byl stejně jako v předchozím případě vybrán obvod XC7A200T.

Výsledky syntézy nástroje Vivado HLS můžeme vidět v tabulce 3.3. Tento nástroj nedokázal provést syntézu jednoho z jader pro rychlou Fourierovu transformaci. Ostatní kódy syntetizoval bez větších problému.

U HLS nástroje Bambu byly výsledky poněkud horší (viz tab. 3.4) než u nástroje Vivado HLS. Taktéž nedokázal provést syntézu u jednoho z jader FFT a také nedokázal provést syntézu pro algoritmus backprop. Co se týče využitých prostředků, tak byl u všech algoritmů výsledný návrh hůře optimalizovaný, a v mnoha případech zabíral několikanásobně větší plochu. Maximální dosažená frekvence byla větší pouze v případě algoritmu merge sort.

Nástroj GAUT (viz tab. 3.5) byl celkově více optimalizován než nástroj Bambu, avšak také nedokázal provést syntézu u stejných souborů. I přes lepší výsledky oproti Bambu HLS, stále nedosahuje takové optimalizace jako Vivado HLS.

Pro nástroj Intel HLS sice vysokoúrovňová syntéza provedena byla (viz tab. 3.6), ale výstup z této syntézy, lze dále syntetizovat pouze v prostředí od Intelu (Quartus Prime). Jelikož je architektura Intelu (Cyclone V) rozdílná od vybrané architektury Xilinx (Artix 7), nelze je jednoduše porovnat.

Tab. 3.3: Dosažené výsledky nástroje Vivado HLS.

Jádro	LUT	FF	BRAM	DSP	Frekvence [MHz]
aes	676	422	2	0	443,2
backprop	6843	11039	17	0	370,5
bfs/bulk	397	380	0	0	319,4
bfs/queue	450	248	1	0	332,4
fft/strided	526	1717	0	0	405,9
fft/transpose	-	-	-	-	-
gemm/blocked	91	663	0	0	589,7
gemm/cubed	162	581	0	0	536,5
kmp	417	264	0	0	255,7
md/grid	1633	2566	0	0	280,4
md/knn	865	2110	0	0	537,6
nw	465	394	0	0	228,5
sort/merge	389	201	2	0	196,4
sort/radix	593	312	0	0	359,2
spmv/crc	133	650	0	0	455,9
spmv/ellpack	126	584	0	0	348,2
stencil2d	136	232	0	3	356,5
stencil3d	548	548	0	6	356,5
viterbi	965	1024	64	0	310,3

Tab. 3.4: Dosažené výsledky nástroje Bambu.

Jádro	LUT	FF	BRAM	DSP	Frekvence [MHz]
aes	4711	2678	0	0	289,8
backprop	-	-	-	-	-
bfs/bulk	1075	680	0	0	290,2
bfs/queue	914	564	0	0	255,6
fft/strided	4644	2691	0	10	139,9
fft/transpose	-	-	-	-	-
gemm/blocked	2968	1863	0	10	139,9
gemm/cubed	2772	1597	0	10	139,9
kmp	1184	788	0	0	251,1
md/grid	14765	8411	0	61	139,6
md/knn	12840	6966	0	61	139,6
nw	2252	1247	0	0	185,9
sort/merge	996	544	0	0	292,1
sort/radix	2727	1740	0	0	257,9
spmv/crc	2778	1424	0	10	139,9
spmv/ellpack	2677	1459	0	10	139,9
stencil2d	731	496	0	3	185,9
stencil3d	2593	1693	0	6	173,5
viterbi	4247	2351	0	0	190,6

Tab. 3.5: Dosažené výsledky nástroje GAUT.

Jádro	LUT	FF	BRAM	DSP	Frekvence [MHz]
aes	4233	2165	0	0	289,872
backprop	-	-	-	-	-
bfs/bulk	953	612	0	0	261,2
bfs/queue	986	520	0	0	277,3
fft/strided	3988	1863	0	7	153,1
fft/transpose	-	-	-	-	-
gemm/blocked	2568	1132	0	11	148,6
gemm/cubed	2763	1703	0	10	151,1
kmp	635	589	0	0	276,8
md/grid	8137	5492	0	23	186,3
md/knn	7235	5318	0	23	193,8
nw	1459	1125	0	0	173,8
sort/merge	723	451	0	0	180,6
sort/radix	2596	1486	0	0	239,5
spmv/crc	2902	1345	0	6	127,5
spmv/ellpack	2568	1354	0	7	151,3
stencil2d	386	536	0	3	210,6
stencil3d	602	654	0	6	199,3
viterbi	3896	2168	0	0	228,7

Tab. 3.6: Dosažené výsledky nástroje Intel HLS.

Jádro	ALM	FF	RAM	DSP	Frekvence [MHz]
aes	73092	134630	700	0	107,3
backprop	-	-	-	-	-
bfs/bulk	5160	11251	46	0	146,8
bfs/queue	6461	14063	61	0	147,6
fft/strided	20054	53606	98	32	140,1
fft/transpose	-	-	-	-	-
gemm/blocked	61961	151536	526	64	125,8
gemm/cubed	5580	14606	34	8	170,3
kmp	8857	16465	78	0	149,4
md/grid	40012	112302	119	105	146,9
md/knn	36169	102758	153	105	140,5
nw	9545	18835	69	0	144,9
sort/merge	990	2103	7	0	105,1
sort/radix	-	-	-	-	-
spmv/crc	5432	14821	37	8	157,7
spmv/ellpack	41321	11045	111045	178	80
stencil2d	8275	17280	68	18	153,4
stencil3d	12801	25115	126	4	145,9
viterbi	16808	44043	213	0	227,1

ZÁVĚR

Cílem bakalářské práce bylo zjistit jaký má vliv návrh v jazyce vysoké úrovně na konečný návrh FPGA obvodů. Tento vliv se odvíjí od nástroje, který je pro HLS použit. V této práci jsou proto tyto nástroje popsány a pro zjištění vlivu a porovnání nástrojů byly navrženy dvě metody, kterými se HLS nástroje testovaly.

První metodou byl zjišťován vliv HLS na výsledný návrh FPGA obvodu. Porovnával se zde návrh v jazyce vysoké úrovně (SystemC), který byl syntetizován pomocí jednoho nástroje HLS (Vivado HLS) s návrhem v jazyce nižší úrovně (Verilog). U většiny testovaných algoritmů byl výsledný návrh číslicového obvodu při použití jazyka vysoké úrovně lépe optimalizovaný a dosahovalo se u něho vyšší frekvence. To se však odrazilo na odhadované spotřebě obvodu, kde LLS nástroj po implementaci odhadl spotřebu u všech HLS návrhů větší, což by mohl být problém pro zařízení napájené z baterie.

Jelikož ne všechny nástroje podporují SystemC tak byl HLS nástroj, který byl porovnáván v první metodě (Vivado HLS), v druhé metodě testován s ostatními vybranými HLS nástroji (Bambu, GAUT, Intel HLS), sadou algoritmů v jazyce C. Podle očekávání, bylo zjištěno, že nástroje, které jsou vyvíjeny na univerzitách (Bambu a GAUT) nejsou tak dobře optimalizované jako Vivado HLS, za kterým stojí firma, která se zabývá vývojem FPGA (Xilinx). Při syntéze s nástroji GAUT a Bambu byla využitá plocha FPGA u všech testovaných algoritmů i několikanásobně větší a maximální frekvence byly až na pár výjimek naopak menší. Intel HLS nelze s nástrojem Vivado HLS porovnat, kvůli rozdíle architektuře cílových platforem.

Z těchto poznatků je vyvozen závěr, že design v jazyce vysoké úrovně dosahuje podobných výsledků jako při návrhu v jazyce nižší úrovně, avšak je lepší používat HLS nástroj od stejného výrobce jako je naše vybraná cílová platforma.

LITERATURA

[1] MEEUS, Wim, Kristof VAN BEECK, Toon GOEDEMÉ, Jan MEEL a Dirk STROOBANDT. An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems* [online]. 2012, **16**(3), 31-51 [cit. 2017-12-06]. DOI: 10.1007/s10617-012-9096-8. ISSN 0929-5585. Dostupné z: <http://link.springer.com/10.1007/s10617-012-9096-8>

[2] NANE, Razvan, Vlad-Mihai SIMA, Christian PILATO, et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* [online]. 2016, **35**(10), 1591-1604 [cit. 2017-12-06]. DOI: 10.1109/TCAD.2015.2513673. ISSN 0278-0070. Dostupné z: <http://ieeexplore.ieee.org/document/7368920/>

[3] COLODRO-CONDE, Carlos, Javier TOLEDO-MOREO, Javier MARTÍNEZ-ALVAREZ, Javier GARRIGÓS-GUERRERO a J. Manuel FERRANDEZ-VICENTE. *Implementing large-kernel 2-D filters using Impulse CoDeveloper*, 2012 Conference on [online]. 1. Karlsruhe, Germany: IEEE, 2012, s. 1-8 [cit. 2017-12-06]. ISBN 978-2-9539987-4-0. Dostupné z: <http://ieeexplore.ieee.org/document/6385358/>

[4] VILLARREAL, Jason, Adrian PARK, Walid NAJJAR a Robert HALSTEAD. Designing Modular Hardware Accelerators in C with ROCCC 2.0. In: *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines* [online]. Charlotte: IEEE, 2010, s. 127-134 [cit. 2017-12-07]. DOI: 10.1109/FCCM.2010.28. ISBN 978-1-4244-7142-3. Dostupné z: <http://ieeexplore.ieee.org/document/5474060/>

[5] JACINTO, H S., Luka DAOUD a Nader RAFLA. High level synthesis using vivado HLS for optimizations of SHA-3. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* [online]. Boston: IEEE, 2017, s. 563-566 [cit. 2017-12-07]. DOI: 10.1109/MWSCAS.2017.8052985. ISBN 978-1-5090-6389-5. Dostupné z: <http://ieeexplore.ieee.org/document/8052985/>

[6] PILATO, Christian a Fabrizio FERRANDI. Bambu: A modular framework for the high level synthesis of memory-intensive applications. In: *2013 23rd International Conference on Field programmable Logic and Applications* [online]. Porto: IEEE, 2013, s. 1-4 [cit. 2017-12-08]. DOI: 10.1109/FPL.2013.6645550. ISBN 978-1-4799-0004-6. ISSN 1946-1488. Dostupné z: <http://ieeexplore.ieee.org/document/6645550/>

[7] COUSSY, Philippe, Cyrille CHAVET, Pierre BOMEL, Dominique HELLER, Eric SENN a Eric MARTIN. GAUT: A High-Level Synthesis Tool for DSP Applications. COUSSY, Philippe a Adam MORAWIEC, ed. *High-Level Synthesis* [online]. Dordrecht: Springer Netherlands, 2008, s. 147-169 [cit. 2017-12-08]. DOI: 10.1007/978-1-4020-8588-8_9. ISBN 978-1-4020-8587-1. Dostupné z: <http://link.springer.com/10.1007/978-1-4020-8588-8_9>

[8] MARTIN, G. a G. SMITH. High-Level Synthesis: Past, Present, and Future. *IEEE Design & Test of Computers* [online]. 2009, **26**(4), 18-25 [cit. 2017-12-09]. DOI: 10.1109/MDT.2009.83. ISSN 0740-7475. Dostupné z: <http://ieeexplore.ieee.org/document/5209959/>

[9] SARKAR, S., S. DABRAL, P.K. TIWARI a R.S. MITRA. Lessons and Experiences with High-Level Synthesis. *IEEE Design & Test of Computers* [online]. 2009, **26**(4), 34-45 [cit. 2017-12-09]. DOI: 10.1109/MDT.2009.84. ISSN 0740-7475. Dostupné z: <http://ieeexplore.ieee.org/document/5209961/>

[10] REAGEN, Brandon, Robert ADOLF, Yakun Sophia SHAO, Gu-Yeon WEI a David BROOKS. MachSuite: Benchmarks for accelerator design and customized architectures. In: *2014 IEEE International Symposium on Workload Characterization (IISWC)* [online]. Raleigh: IEEE, 2014, s. 110-119 [cit. 2017-12-10]. DOI: 10.1109/IISWC.2014.6983050. ISBN 978-1-4799-6454-3. Dostupné z: <http://ieeexplore.ieee.org/document/6983050/>

[11] MORAWIEC, Adam, COUSSY, Philippe, ed. High-Level Synthesis From Algorithm to Digital Circuit. Online-Ausg. Dordrecht: Springer Science + Business Media B.V, 2008. ISBN 9781402085888.

[12] BDTI Certified™ Results for the AutoESL AutoPilot High-Level Synthesis Tool. *BDTi* [online]. Walnut Creek: Berkeley Design Technology [cit. 2017-12-09]. Dostupné z: <https://www.bdti.com/Resources/BenchmarkResults/HLSTCP/AutoPilot>

[13] BDTI Certified™ Results for the Synopsys Synphony C Compiler *BDTi* [online]. Walnut Creek: Berkeley Design Technology [cit. 2017-12-09]. Dostupné z: <https://www.bdti.com/Resources/BenchmarkResults/HLSTCP/Synphony>

[14] High-level synthesis. *The era of HLS !* [online]. [cit. 2017-12-03]. Dostupné z: <https://sites.google.com/site/changan2001/hls>

[15] *Synphony C Compiler: Optimized Hardware from High-Level C/C++* [online]. Mountain View: Synopsys, ©2010. [cit. 2017-12-10]. Dostupné z:

<https://www.synopsys.com/content/dam/synopsys/implementation&
signoff/datasheets/synphony-c-compiler-ds.pdf>

[16] *CYBERWORKBENCH: NEC's High Level Synthesis Solution* [online]. Tokyo: NEC Corporation, 2016 [cit. 2017-12-05]. Dostupné z:
<http://www.nec.com/en/global/prod/cwb/pdf/CWB_Detailed_technical.
pdf>

[17] *Intel® High Level Synthesis Compiler: Reference manual* [online]. Intel Corporation, 2017. [cit. 2017-12-10]. Dostupné z:
<https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/
literature/hb/hls/mnl-hls-reference.pdf>

[18] Intel® HLS Compiler *Intel FPGA and SoC* [online]. ©Intel Corporation, 2017 [cit. 2017-12-10]. Dostupné z:
<https://www.altera.com/products/design-software/high-level-design/
intel-hls-compiler/overview.html>

[19] Customer Success Stories - Mentor Graphics. *Mentor, a Siemens Business, leads in electronic design automation software - Mentor Graphics* [online]. Mentor, a Siemens Business [cit. 08.12.2017]. Dostupné z:
<https://www.mentor.com/hls-lp/success/>

[20] *Stratus High-Level Synthesis: Industry's first high-level synthesis platform for use across your entire SoC design* [online]. Cadence Design Systems, ©2015. [cit. 08.12.2017]. Dostupné z:
<https://www.cadence.com/content/dam/cadence-www/global/en_US/
documents/tools/digital-design-signoff/stratus-ds.pdf>

[21] Impulse CoDeveloper. *Impulse Accelerated Technologies* [online]. Impulse Accelerated Technologies, ©2003 [cit. 2017-12-06]. Dostupné z:
<http://www.impulseaccelerated.com/products.htm>

[22] *Y Explorations: eXCite* [online]. San Jose: Y Explorations, c2013 [cit. 2017-12-07]. Dostupné z:
<http://www.yxi.com/products.php>

[23] *MachSuite: Benchmarks for Accelerator Design and Customized Architectures* [online]. Harvard, 2016 [cit. 2017-12-10]. Dostupné z:
<https://github.com/breagen/MachSuite>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ABEL	Advanced Boolean Expression Language
AES	Advanced Encryption Standard – standard pokročilého šifrování
API	Application Programming Interface – rozhraní pro programování aplikací
ARM	Advanced RISC Machine
ASIC	Application Specific Integrated Circuit – zákaznický integrovaný obvod
BDW	BlueSpec Development Workstation
BSV	BlueSpec System Verilog
CABA	Cycle Accurate Bit Accurate
CSP	Communicating sequential processes
CUPL	Compiler for Universal Programmable Logic
CWB	CyberWorkBench
DSP	Digital Signal Processing – digitální zpracování signálu
FF	Flip-Flop – Klopný obvod
FIFO	First In First Out
FPGA	Field Programable Gate Array – programovatelné hradlové pole
GUI	Graphic User Interface – grafické uživatelské rozhraní
HDL	Hardware Description Language – jazyk pro popis hardwaru
HLS	High-Level Synthesis – vysokoúrovňová syntéza
IDE	Integrated Development Environment – Integrované vývojové prostředí
LIFO	Last In First Out
LUT	LookUp Table – náhledová tabulka
LLS	Low-Level Synthesis – nízkoúrovňová syntéza
LLVM	Low Level Virtual Machine
MIPS	Microprocessor without Interlocked Pipeline Stages – procesor bez automaticky organizované pipeline
PALASM	PAL Assembler
PAL	Programmable Array Logic
PLD	Programable Logic Device – programovatelné logické zařízení
RAM	Random Acces Memory – paměť s náhodným přístupem
RISC	Reduced Instruction Set Computing – procesory s redukovanou instrukční sadou
ROCCC	The Riverside Optimizing Compiler for Configurable Circuits
ROM	Read Only Memory – paměť pouze pro čtení
RTL	Register-Transfer Level
TLM-T	Transaction Level Model with Timing
VHDL	VHSIC Hardware Description Language

VHSIC	Very High Speed Integrated Circuit
XML	eXtensible Markup Language