Analýza a anotace VHDL za účelem parametrizovatelnosti

Zbyněk Křivka, Rudolf Schönecker

krivka@fit.vutbr.cz, schonec@fit.vutbr.cz

zpráva projektu
Virtuální laboratoř aplikace mikroprocesorové techniky
MŠMT 2C06008

http://www.vlam.cz
UNIS spol. s r.o, R&D; ÚTIA AV ČR; VUT Brno, FIT, ÚIFS

A2 Identifikace parametrizovatelnosti MCU architektur, IP bloků pro FPGA, seznam typických IP

Úvod

Identifikace parametrizovatelnosti určuje pozice v systému, ve kterých je systém variabilní v konkrétních nastaveních, které specifikuje uživatel. U systému, který je vytvořen komponentní technologií spočívá parametrizovatelnost ve variabilitě propojení komponent, ze kterých je systém tvořen a v nastavení hodnot vlastností těchto komponent (generiky IP komponent).

Promítneme-li parametrizovatelnost na cílový systém založený na FPGA a produkční cyklus vzniku návrhu pro takovýto systém, nabízí se několik přístupů, jak parametrizaci provést. Na straně software návrhu systému jsou možné parametrizace HDL kódu, ze kterého je systém vysyntetizován, a dále parametrizace netlistů či bitstreamu výsledného návrhu. Možnosti parametrizace ze strany hardware spočívají v možnosti volby konkrétního bitstreamu při nahrávání bitstreamu do FPGA a specifika tohoto nahrávání. Hardware může akceptovat nahrání bitstreamu ve stavu, kdy je systém uvnitř FPGA aktivní (zde se dále hovoří o nutnosti změny celého návrhu nebo změny části návrhu), nebo naopak tuto vlastnost nepodporuje a nahrání bitstreamu je dovoleno pouze ve stavu, kdy je FPGA neaktivní.

Dále budeme parametrizovatelností uvažovat pouze parametrizovatelnost na software bázi. Zde je nutné poznamenat, že již v roce 2006/2007 bylo zjištěno, že konfigurace systému musí být provedena před samotnou syntézou systému, tedy v době, kdy je možné ovlivnit jak propojení komponent, tak i nastavení statických parametrů IP komponent. Modifikovat důležité parametry obvodu v již vysyntetizovaném netlistu nebo výsledném bitstreamu (např.šířky pamětových sběrnic, počáteční hodnoty některých signálů, ...) se ukázalo nereálné vzhledem charakteru syntézy FPGA systému, kdy sofistikované algoritmy syntézy znemožnují jednoznačeně identifikovat konkrétní místo změny těchto parametrů v bitstreamu. Možná je pouze modifikace software v bitstreamu – uvnitř pro software určených datových oken pro vestavěný nebo soft-core procesor či periferiemi, kterou jsou řízeny instrukcemi, pro něž datové okno obsahuje program v podporovaném instrukčním formátu (naplnují se speciálními nástroji, jež jsou dodávány vždy ke konkrétní technologii přímo výrobcem – např. nástroj příkazové řádky Bitinit v případě FPGA desek firmy Xilinx).

Identifikace IP bloků/komponent pro FPGA v rámci HDL kódu (konkrétně VHDL) je poměrně jednoduché, omezujeme-li se pouze na identifikaci informace o rozhraní dané komponenty a hlavně o parametrizovatelném rozhraní komponenty. Podrobnější analýza kódu a funkčnosti komponenty naráží již na klasické překážky v podobě strojového „pochopení“ sémantiky, analyzovat samotnou strukturu kódu není na teoretické bázi problém. Při potřebě praktického přístupu je radno omezit se na vytyčení rozhraní IP komponenty v analyzovaném kódu, obecná analýza kódu je totiž vhledem k rysům jazyka VHDL velice náročná (hovoří se dokonce o tom, že mít k dispozici funkční analyzátor VHDL kódu, který akceptuje 90% VHDL jazyka definovaného standardem, je polovinou předpokladů pro založení společnosti vyvíjející nástroj pro syntézu).

Důkazem předešlého tvrzení může být i fakt, že práce Luboše Lorence na analýze kompletního VHDL byla ve výsledku neúspěšná. Velkou vinu na tom nese nesmyslnost standardu, který ja napsán takovým způsobem, že syntaxe je definována nepoužitelným

způsobem a sémantika není popsána dostatečně jednoznačně. Vytvoření si tak vlastní LALR gramatiky je nadlidský výkon.

Seznam typických IP komponent je možné sestavit na základě různých kritérií. Rozsah co do funkcionality je velice pestrý – počínaje nejjednoduššími IP komponentami, které představují např.tlačítko, LED diodu či LCD display, až po komplexní a vysoce konfigurovatelné komponenty, které zapouzdřují funkcionalitu soft-core procesorů, různých typů sběrnic a rozhraní, matematických algoritmů. Zde záleží, na jaké úrovni budeme dále návrh uvažovat – jednoduché IP komponenty připojitelné k systému, či komplexní IP komponenty dodávané např. s EDK (závislost na použité sběrnici). V rámci projektu VLAM je reálné zaměřit se na IP komponenty generovatelné pomocí nástroje Core Generator (tyto mají VHDL specifikaci entity a blackbox netlist implentaci) a/nebo IP komponenty založené na PicoBlaze soft-core procesoru, připojitelných univerzálním rozhraním k IPB sběrnici (dodává UTIA).

Konfigurace systému

Přístup ke konfiguraci systému, tj. k výběru vhodných komponent a jejich propojení je možné pokrýt třemi základními způsoby – automaticky, poloautomaticky se zásahem poučeného uživatele, a zcela manuální tvorbou propojení systému znalým uživatelem. Poznamenejme, že front-end specifikace systému probíhá v prostředí PE, tj. co se týče formy abstrakce, pohybujeme se na úrovni speciálních beanů v PE, které reprezentují konfiguraci pro konkrétní IP komponenty v FPGA.

Zcela automatická obecná konfigurace systému pro libovolný problém, jehož popis hledáme, není možná již z principu, ohraničíme-li však funkční rozsah systémů, které budou navrhovány, co do potřeby konkrétních zdrojů (v tomto případě IP komponent se specifickou funkčností), jsme schopni stanovit jistou konečnou množinu architektur, na kterých budou s použitím vybraných IP komponent realizovány specifické typy aplikací a řešeny specifické typy problémů. Tímto přístupem lze zjednodušit přístup uživatelům, kteří chtějí využít pouze automatickou specifikaci systému pro základní uvažované třídy problémů (např. výukové lekce, často se opakující návrhy, ...).

Víme-li například, že při řešení algoritmu budou využity čtyři tlačítka pro uživatelský vstup a LCD display pro výstup výsledků algoritmu, máme možnost vytvořit danou architekturu ručně, či využijeme architekturu, která bude tvořena IP komponentami procesoru pro kontrolu přerušení a provádění programu, tlačítek a LCD displeje jako vstupní a výstupní periferie. Pro řešení je možné vybrat každou takovou architekturu z dostupných v množině předpřipravených architektur, která obsahuje potřebné IP komponenty a svou velikostí a parametry splňuje omezení cílové FPGA desky.

Ruční a poloautomatická tvorba FPGA systému vyžaduje, aby uživatel byl znalý nejen významu vlastností konkrétních beanů v PE, ale měl vědomosti také o konfiguraci systému pro FPGA, jež je následně vygenerována (ať už v podobě VHDL zdrojových kódů či projektových/konfiguračních souborů pro syntézní nástroje pracující na vyšší úrovni abstrakce – např. EDK). Minimální znalost výsledných vygenerovaných konfigurací pro použitý syntézní nástroj je nutná alespoň kvůli sledování průběhu syntézy navržené FPGA architektury. Architektura musí splňovat omezení na velikost, použitou frekvenci, časové a další závislosti (constraints), které je třeba dodržet a ne každá syntéza FPGA návrhu je úspěšná, přestože je popis konfigurace systému správný (zde opět narážíme na to, že předpřipravený systém má „prakticky“ zaručen, že jeho syntéza proběhne úspěšně, pro

systém, jehož konfigurace bude libovolným způsobem modifikována, není zaručena ani tato skutečnost).

A2-3 Analýza VHDL za účelem detekce komponent a jejich parametrů

Úvod

Primárním cílem této aktivity bylo ověřit myšlenku detekce rozhraní dodaných IP komponent v předloženém VHDL kódu a její náročnost a použitelnost zjištěných informací pro sestavení znalostní báze dat. Znalostní báze dat by sloužila jednak při vytváření speciálních beanů pro PE, kterými by uživatel zadával konfiguraci IP komponent použitých v návrhu FPGA systému, dále pak při generování HDL návrhu systému z těchto konfiguračních informací.

Studium se v průběhu rozdělilo na dvě části:

1. Ověření analýzy VHDL za účelem detekce komponent a jejich parametru – zvolen byl přístup prototypování analyzátoru kompletního VHDL, který je popsán standardem IEEE 1076-2002 VHDL.
2. Doplňující možnost analýzy projektových souborů EDK, které představují alternativu konfigurace systému s vyšší úrovní abstrakce. Zde byl zvolen přístup prototypování analyzátorů pro jednotlivé podporované formáty, které jsou podrobně popsány v dokumentaci Reference Guides dodávané k nástrojům Xilinx EDK a ISE.

Ověření analýzy VHDL

Cíl

Zde se podařilo ověřit, že hlavní myšlenka parsování je životaschopná, ale kvůli rozsáhlým závislostem v gramatice je kompletní parsing velmi náročný.

Analýza kompletní VHDL

Pro přiblížení - uvážíme-li např. gramatiku pro analýzu jazyka C, setkáme se pouze s jedním shift-reduce konfliktem v podobě jazykové konstrukce if-then-else, který se však řeší implicitní volbou operace shift, která je i „natvrdo zadrátována“ do generátoru překladačů – nástroje bison (nepředpokládá se, že by gramatika jazyka obsahovala shift-reduce konflikty, které by bylo potřeba řešit volbou redukce, autoři bisona mají toto jako TODO). Jazyk VHDL je však natolik složitý z pohledu analýzy syntaxe a gramatika jazyka vystavěna ve specifikaci jazyka („standard VHDL“) je komplikovaná takovým způsobem, že se často setkáváme s konflikty shift-reduce, které je třeba implicitně řešit volbou reduce.

Kompletní analýza VHDL však není nezbytně nutná vzhledem k tomu, které informace o parametrizovatelnosti a rozhraní analyzované IP jsou nezbytně nutné (viz dokument Identifikace parametrizovatelnosti). Jedná se nicméně o podnětnou studii, která může minimálně poskytnout komunitě návrhářů neocenitelný prostředek v podobě ověřené gramatiky VHDL jazyka (více pak v dokumentaci Dr.Lorence), navíc staví horní mez dovedností pro analýzu VHDL kódu podmnožiny standardu (která tudíž nemusí být prototypována pro ověření).

Kompletní popis analýzy a s tím spjatých problém je ve zprávě Dr. Lorence v příloze analyza_vhdl.pdf.

Analýza podmnožiny VHDL

Dostačujícím prostředkem pro získání nezbytných informací může být analýza podmnožiny VHDL jazyka omezující se pouze na rozhraní ENTITY v hlavním HDL dokumentu analyzované IP komponenty. Extrahovány jsou především deklarace parametrů statického nastavení (generics), jejich implicitní hodnoty a typy parametrů.

Generics a umístění sekce ENTITY

Úvod

V dokumentu o identifikaci parametrizovatelnosti [TODO] byla popsána metodika parametrizovatelnosti IP komponent. Parametrizovatelnost může být pouze statická – tj. před procesem syntézy jsou specifikovány hodnoty konstant GENERICS v entitě HDL zdrojového souboru IP komponenty.

Nástroj detekující a analyzující oblasti parametrizovatelnosti ve zdrojových souborech ve formátu HDL může provádět analýzu různé komplexity. Počínaje nejjednodušší variantou – vyhledáváním klíčových slov v textu a zběžnou kontrolou syntaxe definice komponenty, přes náročnější formy analýzy HDL, až po analýzu dokumentu s podporou kompletního HDL.

Volba konkrétní varianty je kompromisem mezi kvalitou a náročností implementace analýzy.

Nabízí se však další způsob v podobě rozšíření jednoduchých metod analýzy o přidání dodatečných informací do HDL souboru, které by bud usnadnily analýzu samotnou nebo dále doplnily kontext či dodaly další sémantiku definice.

Tyto dodatečné informace by měly být vloženy v podobě speciálních komentářů o předepsané syntaxi.

Část gramatiky VHDL pro top-level část popisu

Systém je obvykle popsán jako kolekce komponent. Každá komponenta má množinu portů, které představují rozhraní pro připojení komponenty do systému (dynamická vlastnost) a množinu generických konstant - GENERICS (statická vlastnost), které tvůrce komponenty určil jako volné body, v nichž je možné komponentu při jejím připojování do systému uzpůsobit – parametrizovat.

Entity je ve standardu definována pomocí EBNF následovně:

```
entity_declaration ::=
     entity identifier is
          entity_header
          entity_declarative_part
     [begin entity_statement_part]
     end [entity_simple_name] ;
```

Přičemž hledané **generics** definice můžeme najít v části entity_header:

```
entity_header ::= [formal_generic_clause] [formal_port_clause]
generic_clause ::= generic ( generic_list ) ;
generic_list ::= generic_interface_list
```

Konkrétní příklad dané generics v kodu může vypadat následovně:

```
generic (N : Natural := 2) ;
```

Podporované typy generics

Samotný standard VHDL popisuje syntaxi generics, ale co se týká sémantiky a dalších omezení na povolené typy, atd. zde již potměšile mlčí. Z pohledu analýzy syntézy HDL kódu, který má být podroben syntéze bude plně dostačující podporovat generics pro tyto základní typy:

- číselné typy
- výčtové typy
- std_logic, std_ulogic, bit, boolean, character
- std_logic_vector, bit_vector
- jednoduché číselné podtypy

Rozšíření jazyka VHDL pro tvorbu komponent v PE – Anotace VHDL

Analýza generic

Při analýze syntaxe a sémantiky generics části zdrojového textu ve VHDL lze získat tyto statické informace o parametrizaci komponenty a/nebo návrhu:

- identifikace typu parametru
- identifikace hodnoty parametru
- identifikace komentáře

Rozšíření možností generic

Další informace lze získat pouze bočními informačními kanály, kdy mám dvě možnosti:

- konfigurační soubor například v XML formátu nebo proprietárním formátu jako u nástrojů firmy Xilinx

rozšíření sémantiky komentářů (původně pouze pasivní lexikální jednotka jazyka VHDL) o možnost zapisovat v nich doplňující informace o parametrech či vlastnostech dané parametrizace komponenty.

Počáteční návrh jazyka pro anotaci v komentářích VHDL

Jazyk VHDL sám o sobě neobsahuje potřebné jazykové konstrukce pro vyjádření doplňujících sémantických informací, které by dále vzhledem k uvažovanému použití byly zapotřebí. Jedná se zejména o upřesňující informace o oboru hodnot, jehož mohou příslušné generics nabývat. Tyto informace lze doplnit do zdrojového kódu ve strukturovaném formátu do komentářů k příslušným generics.

Příklad formátu doplňujícího komentáře:

```
--GENCMNT! ENUM(VALUE_1, …, VALUE_N)
--GENCMNT! MIN(VALUE), MAX(VALUE)
```

Architektura znalostní databáze

Báze znalostí využívaná jak Processor Expertem při generování konfiguračních souborů pro FPGA generátor, tak FPGA generátorem, který musí bých schopen řešit implementační detaily návrhu z PE.

Prostředky vztahující se k databázi znalostí:

- XML (PE) – na základě databáze znalostí je schopen PE generovat návrh (PE bude využívat jen část databáze znalostí)
- Anotace (povolené hodnoty, implicitní hodnoty parametrech v GENERIC v případě, že nejsou uvedeny v konfiguračních XML souborech generovaných Processor Expertem) (FPGA generátor) – otázka, zda přímá anotace ve VHDL má smysl při samotném generování VHDL návrhu; spíše se nabízí situace, když bude anotované VHDL hrát klíčovou roli v případě přidávání nové VHDL komponenty do databáze znalostí, která tak bude mít zjednodušenu detekční práci u vlastností, které nejsou rozpoznatelné ze samotného VHDL (uživatel může myslet na využití komponenty v našem vývojovém řetězci už během psaní zdrojového textu ve VHDL a ušetří si tak manuální doplňování automaticky nedetekovaných vlastností).
- FPGA generátor – nástroj, který bude nejsofistikovaněji využívat navrženou databázi znalostí, aby dokázal generovat kvalitní VHDL návrhy, jejichž syntéza dopadne úspěšně a pokud možno i s dobrou náročností na zdroje.

Návrh databázi znalostí:

Inspirací pro vypracování tohoto úkolu byla především dokumentace k nástrojům Xilinx ISE a EDK, dále pak zkušenosti získané při práci na demo projektu pro otestování integrace PE a FPGA.

Seznam vlastností v databázi znalostí:

- podporované architektury
  - typ procesoru
  - typy sběrnic
  - popis struktury architektury s zástupnými místy, kam bude možno připojovat komponenty (tzv. přípojné body architektury)
  - definice přípojných bodů (jaké komponenty či rozhraní mohou být připojeny na daný přípojný bod)
- podporované komponenty
  - v jakých podporovaných architekturách lze komponentu použít
  - seznamy podporovaných parametrů komponenty a jejich povolené hodnoty včetně hodnot implicitních v rámci dané architektury

Implementace znalostní databáze

Jelikož se jedná o velmi náročnou strukturu s velkým množství abstrakcí, tak není vhodné využívat jednoduché relační schéma. Již v prvotních fázích bylo rozhodnuto o objektovém modelu.

Problémem persistence objektového modelu je obtížná slučitelnost s von Neumannovskou architekturou dnešních počítačů, kdy nemám k dispozici objektovou paměť. Daný problém je většinou řešen mezivrstvou mezi programem a operačním systémem (často tzv. objektový virtuální stroj).

V případě jazyků nevyužívajících virtuální stroj je však objektová persistence řešena pomocí více či méně kvalitně implementovaných knihoven třetích stran a zacházení s objekty není tak intuitivní, jak by mohlo být. Původní požadavek využívat pouze jazyk C++ se proto ukázal jako velmi nevhodný.

Samotná persistence objektových dat může být řešena různými způsoby, z nichž ty pro nás podstatné jsou:

- objektové databáze (většinou pouze komerční a drahé, některé databáze jsou zdarma dostupné za omezených licenčních podmínek a výkonnostních limitů (Caché od Intersystems))
- XML (velmi volně strukturovaný formát umožňující pohodlnou imlementaci stromové struktury a křížovými odkazy mezi uzly stromu)
- ukládání do souborů s proprietárním formátem (nevhodné pro naše řešení, málo výkonné, málo čitelné)

Závěr

Součástí této aktivity bylo protypování parseru kompletního VHDL. Z hlediska potřeb projektu VLAM bylo ověřeno, že možnosti částečné analýzy VHDL za účelem detekce komponent a jejich parametrů jsou dostatečné. Extrakce dostupných komponent, jejich rozhraní je možná a to v obou uvažovaných větvích výzkumu rámce této aktivity. Výzkum by se dál měl ubírat směrem ke schopnosti detekce propojení již nalezených komponent a to na podčásti jazyka VHDL definovaného standardem.