Semestrální práce z předmětu 36PJ

Překlad VHDL do HTML

Michal Trs, trsm1@fel.cvut.cz

Zadání

Převaděč zdrojových souborů syntetizovatelného VHDL do HTML. Převaděč kontroluje syntaxi, obarvuje výstup a odkazuje od použití signálů a proměnných na jejich deklaraci. Vstupem je tedy *soubor*.vhd a výstupem *soubor.vhd*.html.

Rozbor řešení

Lexikální analyzátor jsem realizoval tabulkou. Pomocí automatu identifikuje speciální znaky jako jsou závorky, dvojtečka, středník, logické operátory, dále čísla, řetězce, atribut, bit a identifikátor. Pokud je načtený symbol identifikátor, prohledám tabulku klíčových slov a případně vrátím token kw<klicove_slovo>. Dále vracím token eof, pokud je načten konec souboru a token err pokud došlo k chybě (např.: špatný formát čísla, identifikátor začíná číslem, atd...).
Syntaktický analyzátor jsem realizoval funkcemi rekurzivního sestupu. Jedná se o překladovou gramatiku. Pouze v několika málo případech jsem musel použít dědičný atribut, abych dokázal rozhodnout, zda pro načtený identifikátor budu vytvářet v HTML souboru návěští, nebo naopak se na návěští budu odkazovat. Dále jsem potřeboval tabulku Globálních deklarací a tabulku lokálních deklarací, aby při odkazování na proměnné docházelo k zastiňování.

Gramatika

Vycházel jsem z popisu jazyka v notaci EBNF na této adrese http://opensource.ethz.ch/emacs/vhdl93_syntax.html, kterou jsem dále upravil (zjednodušil). Není podporována většina deklarací v entitě, jako např.: port map. Zde uvádím upravenou gramatiku kterou jsem implementoval.

adding_operator ::= **symbSign** | **symbOpAnd**

architecture_body ::= **symbKwArch symbIDENT symbKwOf** **symbIDENT** symbKwIs declaration_part **symbKwBegin** architecture_statement_part **symbKwEnd** [**symbKwArch**] [**symbIDENT**] **symbSemicolon**

architecture_statement_part ::= { concurrent_statement }

assertion_statement ::= **symbKwAssert** expression
[**symbKwReport** expression] [**symbKwSever** expression]
symbSemicolon

attribute_name ::= **symbATRIB**
[**symbBraOp** expression **symbBraCl**]

block_declarative_item ::=type_declaration | subtype_declaration
| constant_declaration | signal_declaration
| variable_declaration | use_clause | component_declaration

case_statement ::= **symbKwCase** expression **symbKwIs**
case_statement_alternative { case_statement_alternative }
symbKwEnd symbKwCase [**symbIDENT**]
symbSemicolon

case_statement_alternative ::= **symbKwWhen** choice **symbChoice**
sequence_of_statements

component_declaration ::= **symbKwComp symbKwIDENT**
[**symbKwIs**] [generic_clause] [port_clause]
symbKwEnd symbKwComp [**symbIDENT**]
symbSemicolon

concurrent_statement ::= process_statement | assertion_statement
| concurrent_statement2

concurrent_statement2 ::= **symbIDENT symbCOLON**
concurrent_statement3 | concurrent_statement4

concurrent_statement3 ::= process_statement | assertion_statement
| generate_statement

concurrent_statement4 ::= name3 signal_assignment_statement

constant_declaration ::= **symbKwConst** name_list **symbColon** name
[**symbVarAss** expression] **symbSemicolon**

context_item ::= Library_clause | Use_clause

declaration_part ::= { block_declarative_item }

design_file ::= Design_unit { Design_unit }

design_unit ::= { Context_item } Library_unit

element_association ::= **symbKwOthers symbChoice** expression

entity_declaration ::= **symbKwEnt symbIDENT symbKwIs**
entity_headerdeclaration_part

[**symbKwBegin** entity_statement_part] **symbKwEnd**
[**symbKwEnt**] [**symbIDENT**] **symbSemicolon**

entity_header ::= [generic_clause] [port_clause]

entity_statement ::= assertion_statement | process_statement

entity_statement_part ::= { entity_statement }

enumeration_type_definition ::= **symbBraOp symbIDENT**
{ **symbCOMMA symIDENT** } **symbBraCl**

expression ::= relation { **symbOpLog** relation }
| relation [**symbOpLogSpec** relation]

factor ::= [**symbKwAbsNot**] primary

generate_statement ::= generation_scheme **symbKwGen**
[declaration_part **symbKwBegin**] { concurrent_statement }
symbKwEnd symbKwGen [**symbIDENT**]
symbSemicolon

generation_scheme ::= **symbKwFor** parameter_specification
| **symbKwIf** expression

generic_clause ::= **symbKwGenic symbBraOp** interface_list
symbBraCl symbSemicolon

choice ::= **symbIDENT** | **symbKwOth**
| **symbSTRINGorVECTOR**

identifier_list ::= **symbIDENT** { **symbCOMMA symbIDENT** }

if_statement ::= **symbKwIf** expression **symbKwThen**
sequence_of_statements { **symbKwElsif** expression
symbKwThen sequence_of_statements }
[**symbKwElse** sequence_of_statements] **symbKwEnd**
symbKwIf [**symbIDENT**] **symbSemicolon**

interface_declaration ::= [**symbKwConst** | **symbKwSignal**
| **symbKwVar**] identifier_list **symbColon** [mode] name
[**symbKwBus**] [**symbVarAss** expression]

interface_list ::= interface_declaration
{ **symbSemicolon** interface_declaration }

library_clause ::= **symbKwLib** name_list **symbSemicolon**

library_unit ::= entity_declaration | architecture_body

literal ::= **symbINT** | **symbREAL**
| **symbSTRINGorVECTOR** | **symbKwNull** | **symbBIT**

mode ::= **symbKwIn** | **symbKwMode**

name ::= name2 | **symbSTRINGorVECTOR**

name_list ::= **symbIDENT** { **symbCOMMA symbIDENT** }

name2 :: = **symbIDENT** name3

name3 ::= [attribute_name | selected_name | slice_name]

null_statement ::= **symbKwNull symbSemicolon**

parameter_specification ::= **symbIDENT symbKwIn** range

port_clause ::= **symbKwPort symbBraOp** interface_list
symbBraCl symbSemicolon

prefix ::= **symbIDENT symbDot** prefix | **symbKwAll**

primary ::= name | literal | **symbBraOp** expression
symbBraCl | element_association

process_declarative_item ::=type_declaration | subtype_declaration
| constant_declaration | variable_declaration | use_clause

process_declarative_part ::={ process_declarative_item }

process_statement ::= [**symbKwPost**] **symbKwProc**
[**symbBraOp** name_list **symbBraCl**] [**symbKwIs**]
process_declarative_part **symbKwBegin**
sequence_of_statements **symbKwEnd** [**symbKwPost**]
symbKwProc [**symbIDENT**] **symbSemicolon**

range ::= simple_expression
[**symbKwDir** simple_expression]

relation ::= shift_expression
[relational_operator shift_expression]

relational_operator ::= **symbOpRel** | **symbSigAss**

selected_name ::= **symbDot** prefix

sequence_of_statements ::={ sequential_statement }

sequential_statement ::= wait_statement | assertion_statement
| if_statement | case_statement | null_statement
| sequential_statement2

sequential_statement2 ::= **symbIDENT symbColon**
sequential_statement3 | sequential_statement4

sequential_statement3 ::= | wait_statement | assertion_statement
| if_statement | case_statement | null_statement

sequential_statement4 ::= name3 | signal_assignment_statement
| variable_assignment_statement

shift_expression ::= simple_expression
[**symbOpShift** simple_expression]

signal_assignment_statement ::= **symbSigAss** waveform
symbSemicolon

signal_declaration ::= **symbKwSignal** name_list **symbColon** name
[**symbKwReg** | **symbKwBus**] [**symbVarAss** expression]
symbSemicolon

simple_expression ::= [**symbSign**] term { adding_operator term }

slice_name ::= symbBraOp range **symbBraCl**

subtype_declaration ::= **symbKwSubtype symbIDENT symbKwIs**
name **symbSemicolon**

term ::= factor { **symbOpMult** factor }

type_declaration ::= **symbKwType symbIDENT**
[**symbKwIs** enumeration_type_definition]
symbSemicolon

use_clause ::= **symbKwUse symbIDENT** selected_name
{**symbCOMMA symbIDENT** selected_name }
symbSemicolon

variable_assignment_statement ::= **symbVarAss** expression
symbSemicolon

variable_declaration ::= [**symbKwShared**] **symbKwVar** name_list
symbColon name [**symbVarAss** expression]
symbSemicolon

wait_statement ::= **symbKwWait** [**symbKwOn** name_list]
[**symbKwUntil** expresion] [**symbKwFor** expresion]
symbSemicolon

waveform ::= expression [**symbKwAfter** expression]
| **symbKwNull** [**symbKwAfter** expression]

Seznam klíčových slov

abs, not, after, use, architekture, assert, begin, bus, case, component, constant ,to, downto, else, elsif, end, entity, for, generate, generic, if, in, is, library, out, inout, buffer, linkage, name, null, of, on, others, port, postponed, process, register, report, return, severity, shared, signal, subtype, then, type, unaffected, until, all, variable, wait, when

Závěr

Překladač jsme implementoval v jazyce Java jako konzolovou aplikaci. Testoval jsem ji na zdrojových souborech komponenty DRAM scheduler z projektu Liberouter. Pro praktické použití by bylo potřeba implementovat celou gramatiku VHDL, aby bylo možné překládat sofistikovanější deklarace a propojení komponent a dále testbenche, které nejsou syntetizovatelné.