HDL IP Cores Search Engine Based on Semantic Web Technologies

Chapter *in* Communications in Computer and Information Science · January 2011

DOI: 10.1007/978-3-642-19325-5_31

HDL IP Cores Search Engine based on Semantic Web Technologies

Vladimir Zdraveski[1], Milos Jovanovik[1], Riste Stojanov[1], Dimitar Trajanov[1]

[1] Faculty of Electrical Engineering and Information Technologies – Skopje, Rugjer Boskovic bb, 1000 Skopje, Republic of Macedonia
vladimir.zdraveski@feit.ukim.edu.mk, milos@feit.ukim.edu.mk, ristes@feit.ukim.edu.mk, dimitar.trajanov@feit.ukim.edu.mk

Abstract. A few years ago, the System on Chip idea grew largely and ‘flooded’ the market of embedded systems. Many System on Chip designers started to write their own HDL components and made them available on the Internet. The idea of searching for a couple of pre-written cores and building your own System on Chip only by connecting them seemed time saving. We’ve developed a system that enables a semantic description of VHDL IP components, allows search of specific components based on the unambiguous semantic description and works with prebuilt VHDL IP cores. We present an application built around the system and focus on the benefits the application user gains during the process of System on Chip design.

Keywords: HDL, Semantic Web, VHDL, System on Chip, Components, Search, Composition.

1 Introduction

Embedded systems give intelligence to many devices that we use in everyday life – they are found in everything from mobile phones and MP3 players, cars and home appliances, to complex controllers. The continuous progress of semiconductor technology has made it possible to implement complex systems on a single chip, which has led to new challenges in design methodologies. System on Chip (SoC) is a complex integrated circuit, or integrated chipset, which combines the major functional elements or subsystems of a complete end product into a single entity.

The design of SoC would not be possible if every design started from scratch. In fact, the design of SoC depends heavily on the reuse of Intellectual Property blocks - which are called “IP Cores”. IP reuse has emerged as a strong trend over the last years and has been one key element in closing what the International Technology Roadmap for Semiconductors calls the “design productivity gap” - the difference between the rate of increase of complexity offered by advancing semiconductor process technology, and the rate of increase in designer productivity offered by advances in design tools and methodologies [1], important to offer ways of enhancing designer productivity - although it has dramatic impacts on that. It also provides a mechanism for design teams to create SoC products that span multiple design disciplines and domains. The availability of both hard (laid-out and characterized) and soft (synthesizable) IP cores from a number of IP vendors allows design teams to drop them into their designs and thus add a required functionality to an integrated SoC. In this sense, the advantages of IP reuse go beyond productivity—it offers both a large reduction in design risk, and also a way for SoC designs to be done that would otherwise be infeasible owing to the length of time it would take to acquire expertise and design IP from scratch.

Soft IP cores are usually written in some Hardware Description Language (HDL) like VHDL [2], Verilog or SystemC and System Verilog. Following the trend for open source development, there is a large number of available open source HDL components. In order to design a complete SoC, one should interconnect many single VHDL components, spending a lot of time on the compatibility analysis. Today's HDL search tools are generally statistic-based, so the process of searching a specific component is quite difficult. For instance, it is not a trivial task to search for an 8–bit counter with 1 clock pin and 1 chip-enable pin. Instead, one should download, open and analyze many HDL projects, before deciding whether the IP core matches or not. In order to solve this problem, there is a need to have a more meaningful description of attributes and the function of the HDL component. One of the ways to accomplish this is to add semantic description to the HDL component. The use of semantic annotation of HDL files gives way to many new and different opportunities for improvement to the storage and search process of HDL search engines.

Semantic information gives the machine the ability to know and decide and do much more of the work than it was doing previously. Instead of storing HDL information simply as a text file, with the use of semantic web technologies the machine can understand more about the kind and interface of a HDL component. That knowledge enables automated search by I/O interfaces, component type, and further composition of an entire SoC by the use of HDL IP cores.

In order to test the idea in practice, we developed a semantic extension of VHDL and designed a system for it. The system has a module for automatic VHDL annotation, a module for manual semantic annotation, annotated semantic data storage, search and composition of HDL IP cores.

2 Related Work

2.1 HDL Repository Web Portals

There are numerous open source HDL code projects and a few web portals (groups, environments) that enable storage and search of HDL projects. One of them is "Open Cores" [3], where existing IP cores can be found and downloaded. The search process requires the user to search only by name or by the type of the IP core needed. Although there are thousands of projects in this repository, it is quite difficult to find a specific IP component that contains specific ports (e.g. an 8-bit buffer).

Another example is the Java optimized processor's group [4], where a project for building a processor from scratch and optimizing it to execute Java instructions is shown. But, here the user faces the same problem: despite the fact that this project contains many IP cores, e.g. 8-bit buffers, finding a specific one is not so straight-forward. To find a specific core, one should search through all the folders and analyze files one at a time, which takes a lot of time and sometimes ends unsuccessfully. There are also many other similar examples, [5][6][7][8][9].

Also, there are some plug-ins for programming environments that enable inserting of pre-made IP cores, such as standard types of memories, buffers, counters, etc. For instance, Xilinx ISE [10] has its own library, which makes it quite user-friendly. But, the same problem of finding a specific component exists.

2.2 Semantic Search Systems

On the other hand, the technologies of the Semantic Web allowed development of novel approaches to data storage and retrieval. A semantic search system is essentially an information retrieval system which employs semantic technologies in order to enhance different parts of the information retrieval process. This is achieved by semantic indexing and annotation of content, query expansion, filtering and ranking the retrieved information [11]. The semantic search also introduces additional possibilities, such as search for online ontology [12], search for online (distributed) knowledgebase, retrieval of facts from the ontology and knowledgebase and question answering.

A recent survey showed that there is a diversity of approaches in semantic search systems, based on the following categories: search goals, scope, ontology encoding, knowledge richness, user input, architecture, and search phrase support [11].

From the viewpoint of search goals, semantic search can be classified into information retrieval, data retrieval [13], question answering [14] or ontology retrieval [12]. The scope of a semantic search can be the Web [15][16][17][18], desktop search [19][20] or domain repositories [21]. The encoding format of the ontologies used can be a proprietary format [22], OWL or RDFS as open standards [13][20], or even some other format [23][24][25]. In order to enhance the semantic search over the traditional, statistical and syntactical search, the researchers used to focus on the use of thesaurus and taxonomy [26][27]. But in recent year, with the development of the semantic web technologies, it is possible to use richer and more complex knowledge structures, such as classes, instances, object properties, relations, axioms, etc. [14]. Based on the user interactions required by the system, the different approaches fall into one of the following categories [11]: simple keyword based entry into text field [18], natural language sentences [14], graphical taxonomy/ontology browsing [23][28], multi-optional specification of search parameters [27], use of a formal ontology query language [29][30], and interactive with explicit user feedback [31].

From the perspective of the architecture of semantic search systems, most of the developed applications are domain-specific intranet or desktop applications, built over semantically annotated data from a certain domain [20][24]. The main reason for this is the most common problem the semantic web researchers face: the lack of semantically annotated data on the Web. Still, there are other types of applications which use different semantic techniques on top of existing standard search engines [18].

3 Overview of HDL IP Cores Search Engine

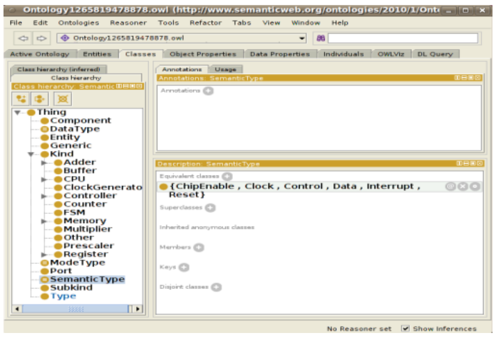
Our approach is to use information retrieval for desktop search over a local, domain ontology-based knowledgebase. We use an OWL domain ontology and an RDF [33] knowledgebase. The documents needed for the application (HDL IP cores) are semantically annotated with concepts from the domain ontology. The architecture of the application puts it in the category of stand-alone web applications and uses its own data repository.

3.1 HDL Ontology

For the purpose of the system, we designed the HDL ontology. Although we would present this ontology from a VHDL perspective, it can be used for classification of any kind of hardware units, chips, etc.

There are specifications about VHDL components and many classes that enable quite original and intuitive classification of different, commonly used VHDL components, written by different authors. Furthermore, there are some predicates and relations that could be used to specify the hierarchy in the RDF description.

The HDL ontology was designed using the Protégé editor [32], shown in Fig. 1. The ontology is used to classify and annotate all of the VHDL components in order to store the details of the users' source code into the system.



Ontology1265819478878.owl (http://www.semanticweb.org/ontologies/2010/1/Ont
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
Ontology1265819478878.owl
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Semantic
Thing
Component
DataType
Entity
Generic
Kind
Adder
Buffer
CPU
ClockGenerato
Controller
Counter
FSM
Memory
Multiplier
Other
Prescaler
Register
ModeType
Port
SemanticType
Subkind
Type
Annotations Usage
Annotations: SemanticType
Annotations
Description: SemanticType
Equivalent classes
{ChipEnable , Clock , Control , Data , Interrupt , Reset}
Superclasses
Inherited anonymous classes
Members
Keys
Disjoint classes
No Reasoner set
Show Inferences

Fig. 1. VHDL ontology in Protégé.
A component could be classified as *Counter*, *Register*, *CPU*, etc. There is information about mode type, data type and semantic type of all the ports. The semantic type describes whether the pin/port is *data*, *control*, *enable*, etc.

The ontology covers relations and classes inside the VHDL code. It allows description of the entity, its ports, their data type, length and mode. A generic section is also considered. In most of the classes the "*Other*" class was nested in order to classify all uncovered hardware components.

Besides VHDL mapping, the ontology also includes additional metadata. There is a semantic kind of the component (to specify whether it is an Adder, a Buffer, a CPU, etc.), author info and frequency specification. Ports are described with the *SemanticType* property, which allows the user to assign semantic meaning to a port. Thus, the user can define the port as *data*, *control*, *enable*, *clock*, etc. These additional information gives novelty to the storage and search engines for HDL components, and adds benefit for the end user. By semantically annotating the different components of a VHDL solution, the user can search for a component that matches some specific pattern, at the level of ports and pins interface, kind and working frequency.

4 Solution Description

As shown in Fig. 2, the HDL IP cores search system consists of a presentation layer (developed using JSP technology), a business layer (developed in Java) and a Jena-based data storage [34][35].

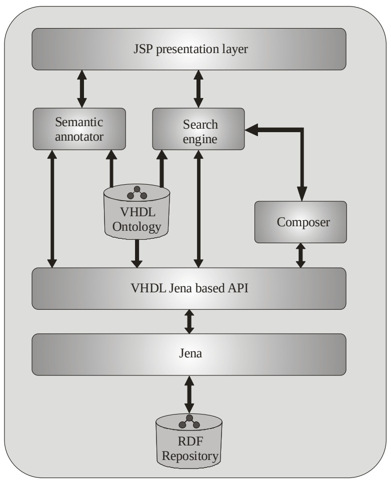
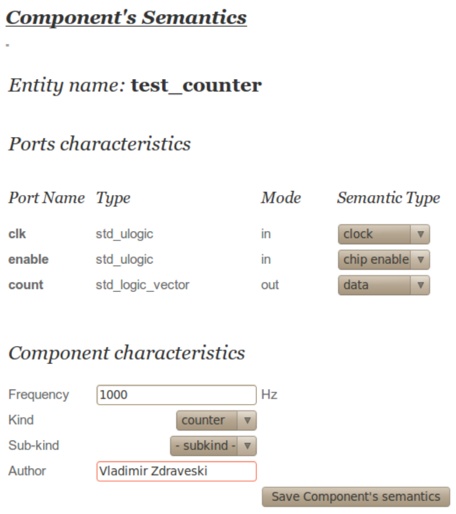


Fig. 2. System architecture.



Component's Semantics

Entity name: **test_counter**

Ports characteristics

Port Name	*Type*	*Mode*	*Semantic Type*
clk	std_ulogic	in	clock
enable	std_ulogic	in	chip enable
count	std_logic_vector	out	data

Component characteristics

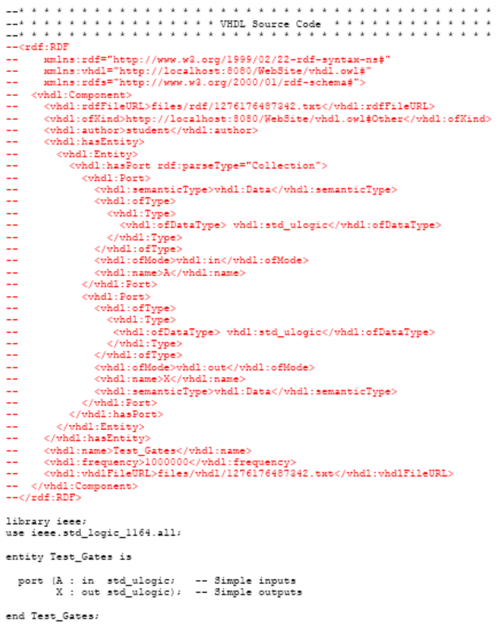
Frequency	1000 Hz
Kind	counter
Sub-kind	- subkind -
Author	Vladimir Zdraveski

Fig. 3. Port semantics.

The business tier includes the semantic annotator, the search engine and the composition engine. The core of the system is the Jena Framework, a Java framework which allows the use of the technologies of the Semantic Web. We also use the Jena repository, which serves for data storage.

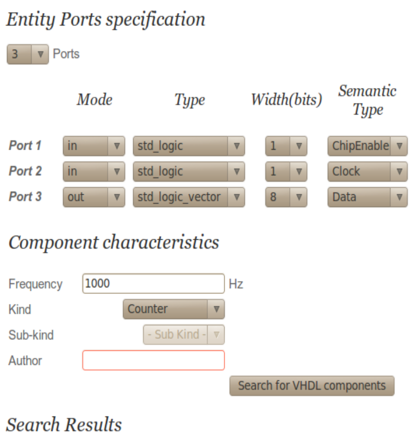
4.1 Semantic Annotator

This module is used for uploading VHDL files on the server. The server parses the files, extracts the needed HDL information of the component and allows adding an additional semantic description (Fig. 3). We modified the Hardware-Vhdl-Parser [36] so that it converts the VHDL entity into its appropriate RDF representation.



```
--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
--* * * * * * * * * * * * * * * * * VHDL Source Code  * * * * * * * * * * * * * *
--* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
--<rdf:RDF
--    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
--    xmlns:vhdl="http://localhost:8080/WebSite/vhdl.owl#"
--    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
--  <vhdl:Component>
--    <vhdl:rdfFileURL>files/rdf/1276176487342.txt</vhdl:rdfFileURL>
--    <vhdl:ofKind>http://localhost:8080/WebSite/vhdl.owl#Other</vhdl:ofKind>
--    <vhdl:author>student</vhdl:author>
--    <vhdl:hasEntity>
--      <vhdl:Entity>
--        <vhdl:hasPort rdf:parseType="Collection">
--          <vhdl:Port>
--            <vhdl:semanticType>vhdl:Data</vhdl:semanticType>
--            <vhdl:ofType>
--              <vhdl:Type>
--                <vhdl:ofDataType> vhdl:std_ulogic</vhdl:ofDataType>
--              </vhdl:Type>
--            </vhdl:ofType>
--            <vhdl:ofMode>vhdl:in</vhdl:ofMode>
--            <vhdl:name>A</vhdl:name>
--          </vhdl:Port>
--          <vhdl:Port>
--            <vhdl:ofType>
--              <vhdl:Type>
--               <vhdl:ofDataType> vhdl:std_ulogic</vhdl:ofDataType>
--              </vhdl:Type>
--            </vhdl:ofType>
--            <vhdl:ofMode>vhdl:out</vhdl:ofMode>
--            <vhdl:name>X</vhdl:name>
--            <vhdl:semanticType>vhdl:Data</vhdl:semanticType>
--          </vhdl:Port>
--        </vhdl:hasPort>
--      </vhdl:Entity>
--    </vhdl:hasEntity>
--    <vhdl:name>Test_Gates</vhdl:name>
--    <vhdl:frequency>1000000</vhdl:frequency>
--    <vhdl:vhdlFileURL>files/vhdl/1276176487342.txt</vhdl:vhdlFileURL>
--  </vhdl:Component>
--</rdf:RDF>

library ieee;
use ieee.std_logic_1164.all;

entity Test_Gates is

  port (A : in  std_ulogic;   -- Simple inputs
        X : out std_ulogic);  -- Simple outputs

end Test_Gates;
```

Fig. 4. Nested RDF description.



Entity Ports specification

3 Ports

	Mode	*Type*	*Width(bits)*	*Semantic Type*
Port 1	in	std_logic	1	ChipEnable
Port 2	in	std_logic	1	Clock
Port 3	out	std_logic_vector	8	Data

Component characteristics

Frequency	1000 Hz
Kind	Counter
Sub-kind	- Sub Kind -
Author	

Fig. 5. Search for a VHDL component.

In order to create self contained, semantically described VHDL documents, this module allows embedding of RDF code directly into the VHDL file, as a VHDL comment. If such a comment exists in the loaded VHDL file, the application reads it directly and doesn't show the form shown in Fig. 3. An example of a nested RDF description within a standard VHDL file is shown in Fig. 4.

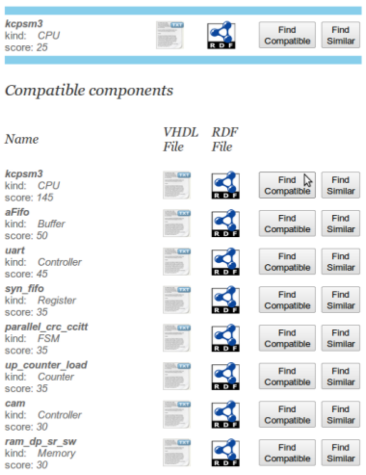
4.2 Repository

The data is stored with the use of the Jena Framework libraries. The Jena Framework stores the data in a graph-like structure, instead of the common database approach which uses strictly formed tables. This graph-like structure is commonly known as an RDF Repository. On top of the Jena Framework we wrote our own API, that provides the functionality according to the VHDL structure. The API provides ontology and data access and is used by the upper layers of the system.

4.3 Search Engine

The search engine is a Java application that uses the Jena API for querying the RDF repository. It is at this point where the semantic annotations of the VHDL components are used to increase the effectiveness over the classic ways of search and retrieval, from the aspect of precision. The search is made by matching the semantic concepts specified in the user request and the semantic annotations of the available components from the RDF repository of the system. We use a semantic based comparison algorithm, which checks for port, frequency and component class matching. We assigned different weighs for different properties, and the final matching score is a sum of all matching weighs (the numbers below the component names in Fig. 6 represent the matching score).

The search form is shown in Fig. 5, where a user needs to simply specify the type, frequency and port interface of the needed VHDL component. The results are listed as shown in Fig. 6. An AJAX add-on gives a review of the located component (Fig. 7). The buttons on the right side allow further search for similar components, based on the current component. The "Find Similar" button puts the current component on top and starts a search for similar components.

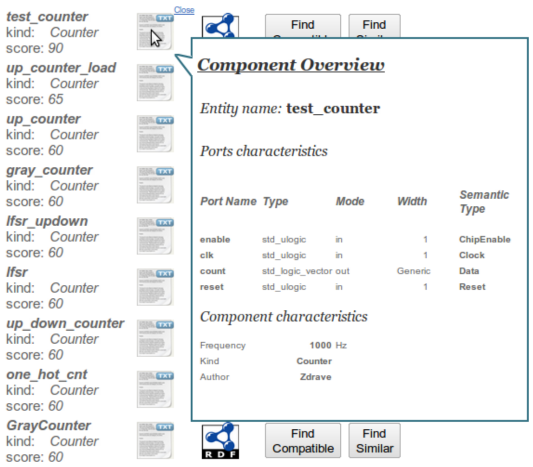


kcpsm3
kind: *CPU*
score: *25*

Compatible components

Name	VHDL File	RDF File		
kcpsm3 kind: *CPU* score: *145*			Find Compatible	Find Similar
aFifo kind: *Buffer* score: *50*			Find Compatible	Find Similar
uart kind: *Controller* score: *45*			Find Compatible	Find Similar
syn_fifo kind: *Register* score: *35*			Find Compatible	Find Similar
parallel_crc_ccitt kind: *FSM* score: *35*			Find Compatible	Find Similar
up_counter_load kind: *Counter* score: *35*			Find Compatible	Find Similar
cam kind: *Controller* score: *30*			Find Compatible	Find Similar
ram_dp_sr_sw kind: *Memory* score: *30*			Find Compatible	Find Similar

Fig. 6. Compatible components.



test_counter
kind: *Counter*
score: *90*

up_counter_load
kind: *Counter*
score: *65*

up_counter
kind: *Counter*
score: *60*

gray_counter
kind: *Counter*
score: *60*

lfsr_updown
kind: *Counter*
score: *60*

lfsr
kind: *Counter*
score: *60*

up_down_counter
kind: *Counter*
score: *60*

one_hot_cnt
kind: *Counter*
score: *60*

GrayCounter
kind: *Counter*
score: *60*

Component Overview

Entity name: **test_counter**

Ports characteristics

Port Name	Type	Mode	Width	Semantic Type
enable	std_ulogic	in	1	ChipEnable
clk	std_ulogic	in	1	Clock
count	std_logic_vector	out	Generic	Data
reset	std_ulogic	in	1	Reset

Component characteristics

Frequency	1000 Hz
Kind	Counter
Author	Zdrave

Fig. 7. AJAX component preview.

4.4 Composer

This module is a base for making a composition of IP cores. The composer module API contains functions for compatibility check, which means that compatible components can be located. Finding a compatible component ("Find Compatible" button in Fig. 6) means finding components that can be connected to a chosen one, for instance, an 8-bit output port is compatible with an 8-bit input port. An example of usage is shown in Fig. 6, where compatible components for the PicoBlaze CPUkcpsm3 are listed (e.g. ram_dp_sr_sw RAM component at the bottom). This is the first step of creating an entire SoC composition by the use of IP cores.

4.5 Advantages and System Usability

When searching through existing HDL portals (aforementioned in the related work section) the result is a folder with the full HDL project, containing many files. In contrast, our HDL IP cores search engine gives a single HDL file that represents a specific component as a result. However, despite this difference, these two approaches (file-based and project-based) are applicable to different problems, so they would and should exist concurrently.

Another advantage of the proposed system is that the component classification is based on an OWL ontology, which enables knowledge sharing and easy resolution of ambiguity. There are common methods for merging semantic data from different ontologies, which makes the HDL IP cores system really scalable and easy to maintain and improve.

Unique features of the HDL IP cores system, which are a direct result of the usage of semantic technologies, are the effortless retrieval of similar components and the ability to search for compatible components. As shown in Fig. 5, Fig. 6 and Fig. 7, the HDL IP cores system enables search “by port” and gives reliable ranking of the similar and compatible components available in the repository. Although we tested the HDL IP cores system with a relatively small set of VHDL examples, we find the testing reliable because of the completeness of the test set. Namely, there are memories, controllers, buffers, flip-flops, counters, registers, multiplexers, coders, decoders, parity generators, etc. in the system. The results from the compatibility search were obvious and expected. For instance, a CPU compatibility search ranks memories higher, and a coder compatibility search places decoders at the top of the result list. An IP core compatibility feature is an approach that allows SoC design via browse through search results.

5 Conclusion and Future Work

The system and the application we designed and developed are intended to demonstrate the ability of the semantic web technologies for building a more precise search engine for VHDL components. The same principle can be used for describing and searching specific chip-products, too. Such a system can be easily implemented within the hardware producer’s web sites or a web market engine.

In general, the improvements which our system offers are quite a large step to a faster and smarter way of storing and searching data. It is described as a VHDL tool here, but it easily applies to any hardware or software, class-based programming code or product specification.

Our future plans include an extension of the current semantic description of the port, with provision of information about whether the port is buffered or not. This feature will introduce the ability to build a complete system automatically. The application itself would be able to decide whether a buffer component is needed or not and find some of the available components from the repository.

With this feature, we will be able to develop a system that automatically composes a logical block of components solely from the user specifications. The user will be required to define the needed inputs and outputs of the circuit, and the system will be able to compose a logical block which consists of semantically annotated VHDL components from the repository, components which can be connected together in a way that satisfies the user's specifications. This is a problem similar to the problem of automatic composition of semantic web services, and our intention is to apply these concepts to composition of IP cores.

References

1. International technology roadmap for semiconductors, Design, 2007 edition, http://www.itrs.net/Links/2007ITRS/2007_Chapters/2007_Design.pdf,
2. VHDL – Very High Speed IC Hardware Description Language , http://www.vhdl.org/
3. Open Cores – web portal, http://opencores.org/
4. Java optimized processor - group , http://tech.groups.yahoo.com/group/java-processor/
5. IP supermarket – web portal, http://www.ipsupermarket.com/index.php
6. Infineon – web portal, http://www.ipsupermarket.com/index.php
7. Lattice – web portal, http://www.latticesemi.com/
8. Chip Estimate – web portal, http://www.chipestimate.com/
9. Design & Reuse – web portal, http://www.design-reuse.com/
10. Xilinx ISE – HDL programming environment , http://www.xilinx.com/
11. Strasunskas, D. and Tomassen, S.L. *On Variety of Semantic Search Systems and Their Evaluation Methods*. Proceedings of International Conference on Information Management and Evaluation, University of Cape Town, South Africa, 25-26 March 2010, Academic Conferences Publishing, pp. 380-387.
12. Pan, J.Z., Thomas, E. and Sleeman, D. (2006) "Ontosearch2: Searching and querying web ontologies", In Proc. of the IADIS International Conference, pp 211-218.
13. Guha, R., McCool, R. and Miller, E. (2003) "Semantic search", In Proc. of WWW 2003, pp 700-709.
14. Lopez, V., Uren, V., Motta, E. and Pasin, M. (2007) "AquaLog: An ontology-driven question answering system for organizational semantic intranets", *Web Semantics*, Vol 5, No. 2, pp 72-105.
15. Stojanovic, N., Studer, R., and Stojanovic, L. (2003) "An approach for the ranking of query results in the Semantic Web", In ISWC 2003, LNCS 2870, Springer-Verlag, pp 500-516.
16. Rocha, C., Schwabe, D. and de Aragao, M. (2004) "A hybrid approach for searching in the semantic web", In Proc. of WWW 2004, ACM Press, pp 374-383.
17. Corby, O., Dieng-Kuntz, R., Faron-Zucker, C., Gandon, F.L. (2006) "Searching the Semantic Web: Approximate Query Processing Based on Ontologies", *IEEE Intelligent Systems*, Vol 21, No. 1, pp 20-27.
18. Tomassen, S.L., and Strasunskas, D. (2009) "A semiotics-driven approach to Web search: analysis of its sensitivity to ontology quality and search tasks", In Proc. of iiWAS'2009, ACM.
19. Kiryakov, A., Popov, B., Terziev, I., Manov, D. and Ognyanoff, D. (2004) "Semantic annotation, indexing, and retrieval", *Journal of Web Semantics* Vol 2, No. 1, pp 49–79.

20. Chirita, P.-A., Costache, S., Nejdl, W. and Paiu, R. (2006) "Beagle++: Semantically enhanced searching and ranking on the desktop", ESWC 2006, LNCS 4011, pp 348-362.
21. Castells, P., Fernandez, M., and Vallet, D. (2007) "An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval". *IEEE TKDE* 19(2), pp 261-272.
22. Amaral, C., Laurent, D., Martins, A., Mendes, A. and Pinto, C. (2004) "Design and Implementation of a Semantic Search Engine for Portuguese", In Proc. LREC 2004, Vol 1, pp 247–250.
23. Brasethvik, T. (2004) *Conceptual modelling for domain specific document description and retrieval- An approach to semantic document modelling*. PhD thesis, NTNU, Trondheim, Norway, 2004.
24. Zhang, L., Yu, Y., Zhou, J., Lin, Ch. and Yang, Y. (2005) "An enhanced model for searching in semantic portals", In WWW 2005, pp 453-462.
25. Burton-Jones, A., Storey, V.C., Sugumaran, V. and Purao, S. (2003) "A heuristic-based methodology for semantic augmentation of user queries on the web", ER 2003, LNCS 2813, pp 476–489.
26. Ciorascu, C., Ciorascu, I., and Stoffel, K. (2003) "knOWLer - ontological support for information retrieval systems", In Proc. of SIGIR 2003 Conference, Workshop on Semantic Web.
27. Aitken, S. and Reid, S. (2000) "Evaluation of an ontology-based information retrieval tool", In Proc. of Workshop on the Applications of Ontologies and Problem-Solving Methods, ECAI 2000, Berlin.
28. Suomela, S. and Kekalainen, J. (2005) "Ontology as a search-tool: A study of real user's query formulation with and without conceptual support", ECIR'2005, LNCS 3408, Springer-Verlag, pp 315-329.
29. Blacoe, I., Palmisano, I., Tamma, V. and Iannone, L. (2008) "QuestSemantics - Intelligent Search and Retrieval of Business Knowledge", Frontiers in Artificial Intelligence and Applications, Vol. 178, pp 648-652.
30. Wang, H., Zhang, K., Liu, Q., Tran, T., and Yu, Y. (2008) "Q2Semantic: A Lightweight Keyword Interface to Semantic Search", ESWC 2008, LNCS 5021, Springer-Verlag, pp 584-598.
31. Nagypal, G. (2007) *Possibly imperfect ontologies for effective information retrieval*. PhD thesis, University of Karlsruhe.
32. Protégé – semantic data editor, RDF, OWL…, Stanford Center for Biomedical Informatics Research, http://protege.stanford.edu/, 2010
33. RDF, Resource Description Framework, http://www.w3.org/RDF/, 2010
34. Jena – A semantic web, java framework,
Official API documentation and examples for Jena libraries, http://jena.sourceforge.net/, 2010
35. Jena TDB storage, http://openjena.org/wiki/TDB, 2010
36. Vhdl-Parser,http://search.cpan.org/~gslondon/Hardware-Vhdl-Parser-0.12/Parser.pm, 2000