Knowledge-Guided Methodology for Third-Party Soft IP Analysis

Bhanu Singh, Arunprasath Shankar,
Francis Wolff, Daniel Weyer, Christos Papachristou
Case Western Reserve University
Email: {bps36, axs918}@case.edu

Bhanu Negi
Marvell Semiconductor
Email: {bhanu}@marvell.com

***Abstract*—In System-on-Chip designs, third party IP reuse is prevalent as it increases productivity and reduces time-to-market. These IPs can be classified as untrusted designs since the user has no insight into IP verification or quality control process. In practice, it is generally assumed that the IP has been functionally validated by developers and thorough verification at user end is not performed.**

In the current state-of-the-art, lint tools are primarily used to determine IP design quality. These tools pinpoint design issues by performing static analysis of RTL code but have a limitation that they do not perform behavioral analysis. In this paper, we present a knowledge-guided methodology, which identifies RTL behavior by finding correspondences with a knowledge base of previously analysed trusted designs. In comparison to existing techniques, our approach uses combination of static and dynamic analysis techniques to better approximate design behavior. We tested our methodology by analysing several IEEE-754 floating point soft IPs. We define identification coverage and confidence factor metric to quantify our IP analysis results.

***Keywords*-RTL analysis; Ontology; Knowledge base; Expert systems; RTL Lint**

I. Introduction

The integration of third-party IPs is a common practice in system-on-chip designs. This allows semiconductor companies to focus their R&D effort on developing their differentiating technology and use third-party IPs for standard-based IPs, which provide minimal differentiation.

A soft intellectual property (Soft IP) core is a reusable unit of logic, which can be synthesized to logic gates. In recent years, the task of verifying an IP against its specification is becoming daunting due to increasing design complexity. In today's global economy, IPs are being designed and verified at different geographic locations. The multi-site development of IPs increases the risk to design quality.

Verification is a quality control process, which is used to evaluate whether a design complies with its specifications. The current verification techniques can be broadly classified into simulation or formal methods. These techniques require building a complex verification environment, which is a time consuming process. In a typical project, 70% of the development time is devoted to verification process [1]. The productivity gap affecting verification means that thorough testing of third party IPs is generally not done. The common practice is to perform basic integration tests and RTL lint checks to flag any IP design issues [2]. However, the importance of thorough design analysis is not overstated. IP design issues, if discovered later than RTL sign-off stage increase the design turn around time and in worst case result in chip re-spins.

In this paper, we propose a knowledge-guided IP analysis methodology, which aids engineers in mitigating risk of third-party IP reuse. In comparison to existing techniques, we perform IP analysis by finding correspondences with a RTL knowledge base (KB) of trusted designs. Our validation approach complements existing IP validation techniques and acts as a lint tool with a behavioral analysis capability. In current practice, an engineer who is analysing a third-party IP has to reverse engineer IP design behavior by manually reviewing specifications and RTL code. The application of our work is to assist an engineer in reverse engineering the design behavior by performing RTL behavioral annotations.

II. Background and Related Work

In general, static analysis (SA) based techniques are used to identify possible bugs in code as well as indicate how closely certain segments of code match specifications. In theoretical computer science, through Rice's theorem and the undecidability of the Halting problem, it has been proved that finding any kind of violation of a specification on the final result of a program is undecidable [3]. The only way to determine how a program will behave is to perform exhaustive simulations. In any case, being forewarned about a bug is valuable as it gives immediate feedback about what may be wrong in the implementation [4]. In our approach, we address the limitations of SA techniques by performing evaluation of code fragments to better approximate design behavior. We are not doing design verification, in the sense, of finding bugs in the design. We are validating that the design meets the specifications by levels of confidence. The individual elements of our technique can be roughly related to work in following areas: (a) reverse engineering, (b) IP quality evaluation, and (c) application of KB systems in hardware design.

Reverse engineering has been an active topic of research for many years. Most of the techniques proposed so far are for a gate-level netlist [5] [6]. Our approach identifies RTL behavior, which is at higher abstraction level than netlist level.

Various approaches have been proposed for IP quality evaluation [7] [8]. Quality metrics have been proposed in the past such as Quality Intellectual Property (QIP) by virtual socket interface alliance (VSIA) [7]. The VSIA determined four factors that determine overall quality of IP: authoring, verification, the maturity of the IP , and the capabilities of the provider. Our approach provides additional metrics to quantify IP design quality.

In SoC design, RTL lint tools are primarily used for ensuring IP design quality. Various commercial tools are available, which check design issues with respect to DFT (design for testability), synthesis, naming conventions and design style [2]. The basic principle used in these tools is static analysis of RTL code. These tools do not perform RTL behavioral analysis and lack the ability to add new user-defined rule templates.

KB systems based on artificial intelligence (AI) technology have been proposed in the past for fast prototyping of hardware designs and for hardware-software partitioning [9] [10]. IBM estimates that it has saved more than \$100 million during the last decade in direct development costs and reduced time to market by using AI technology for the verification of its processors [11]. Our validation approach is also an AI based technique but the application of our work is in RTL analysis without performing exhaustive simulations.

III. RTL analysis Methodology

The main elements of our system are: (a) RTL KB of trusted designs (b) property based model for each design domain, and (c) RTL analysis rule base. Figure1 provides an overview of our approach. The expert system tool that we have used is the C Language Integrated Production System (CLIPS), originally developed by NASA [12] [13]. In what follows, we describe each element of our technique.

A. RTL KB

IP vendors deliver soft IP RTL database as a set of files, each of which generally corresponds to a design module. In our approach, we do a semantically equivalent translation of RTL code into CLIPS facts, which we term as RTL-CLIPS. The RTL-CLIPS preserves the RTL code structure and enables the rule-base to use code semantics to infer a function or generate some high level fact, *e.g.*, to identify a process as an FSM and list its state transition graph. The RTL KB contains a database of previously analysed trusted designs in RTL-CLIPS format.

CLIPS uses a forward-chaining inference strategy based on the Rete pattern-matching algorithm [12]. The CLIPS rules are written by creating a pattern using a set of RTL-CLIPS facts. When a match is achieved, the rule "fires" and an inference is made by asserting new facts.

TABLE I: Examples of HDL represented in Knowledge Base

VHDL Element	CLIPS Template
VARIABLE x;	(var (id gen1) (text "x") (class variable))
z := x + y;	(stmt (cat MATH +) (ins gen1 gen2) (outs gen3))
PROCESS (x)	(stmt (cat BLOCK CONC-SEQ) (ins gen1))
BEGIN stmt4; stmt5; END;	(stmt (cat BLOCK) (stmts gen4 gen5))

1) Basic Constructs: Table I gives some examples of VHDL language elements and their CLIPS templates. We define two kinds of templates, var for data and $stmt$ for commands. Each of these templates contains a slot (the CLIPS term for property) for an id and a slot for a parent id. Each fact, an instance of these templates, has a unique id, and each has a parent, to maintain the block structure of the VHDL code. For ids we use arbitrary generated labels of the form: $gen1$, $gen2$, etc. Var templates contain slots for $text$, the variable name from the VHDL code, and $class$, to distinguish signals from variables.

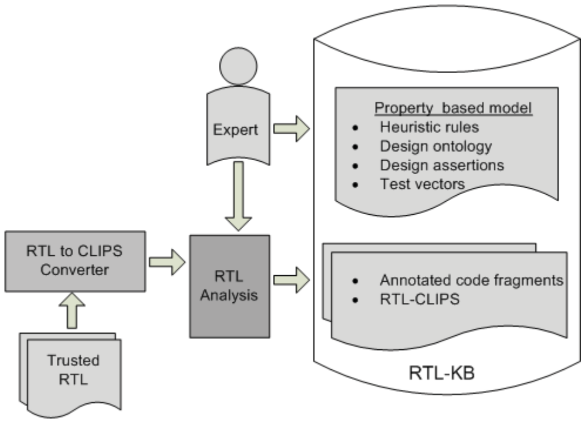
$Stmt$ templates include a multislot cat, for category. A CLIPS slot holds one value whereas a multislot holds an ordered list of values. We define the category as a path through a tree of commands. For example, "MATH +" for addition. $Stmt$ templates also include multislots for ins and $outs$, for ports. In general, the contents of a code block are known by the references to their parent; however for blocks of sequential statements, we also define a $stmts$ multislot in the parent, to give the order of execution. The parent is a $stmt$ template in the category "BLOCK" which broadly includes the VHDL block statements.

This representation is equivalent to a data flow diagram, where each $stmt$ is a node and each var is a hyperedge. To simplify static analysis of code we require that every edge has a source and sink; therefore, we define statement categories for constants and ports. Figure 2 illustrates the relationship of these CLIPS templates and data flow diagrams.

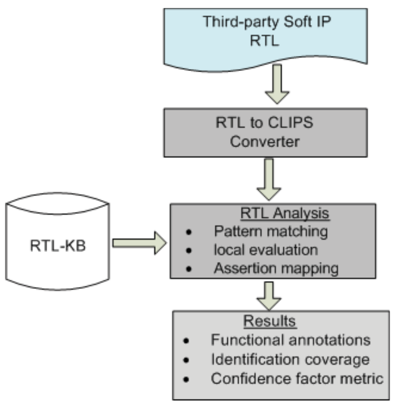
B. Property-based model

Our technique is applicable to soft IP designs, which conform to a standard specification. Standards are published documents that are universally understood to ensure compatible functionality and interoperability. Standard compliant IPs have correspondences in design behavior. For example, every floating point unit (FPU) IP compliant to IEEE-754 standard supports following rounding modes: "round to zero", "round up", "round down" and "round to nearest even" [14]. In our approach, KB has a property based model (PBM) for standard based IPs. A property is a partial specification which is some element of the expected design behavior. At the highest abstraction level, the PBM for a design is represented by its ontology. An ontology is defined as a knowledge representation model to explicitly represent a domain by defining its concepts and various relationships among the concepts [15] [16]. For an IP, the hierarchical functional decomposition of design behavior results in ontological concepts corresponding to behavioral functions. Figure 3 is an example of a fragment of ontology for FPU domain. The FPU ontology was described to the protégé tool and visualized through OntoViz tab [17].

In the ontology, each high level concept is defined in terms of lower level concepts using relationships, for example "has-function", "has-subfunction", "has-operation" and "has-feature". The relationships create a conceptual network, which corresponds to abstract behavioral model for the design. In our ontology representation scheme, functions implement a standalone logic while subfunctions are used by other functions to implement bit-level operations. For example, floating-point



(a) RTL-KB generation



(b) RTL-KB application

Fig. 1: RTL Analysis overview

```
(stmt (cat PORT IN) (outs gen1))
(var (id gen1) (text "A"))
(stmt (cat CONST "5") (outs gen2))
(var (id gen2) (text "B"))
(stmt (cat PORT IN) (outs gen3))
(var (id gen3) (text "C"))
(stmt (cat CONST "7") (outs gen4))
(var (id gen4) (text "D"))
(stmt (cat MATH *) (ins gen1 gen2) (outs gen5))
(var (id gen5) (text "E"))
(stmt (cat MATH *) (ins gen3 gen4) (outs gen6))
(var (id gen6) (text "F"))
(stmt (cat MATH +) (ins gen5 gen6) (outs gen7))
(var (id gen7) (text "G"))
(stmt (cat PORT OUT) (ins gen7))
```

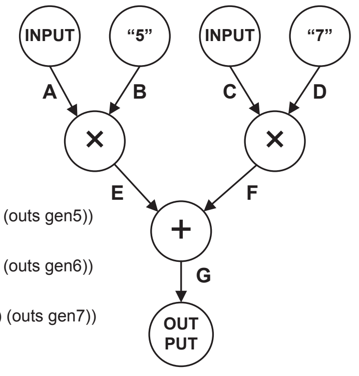


Fig. 2: RTL-CLIPS Facts and Equivalent Data Flow Diagram

add "FPADD" is a FPU function and "rounding" is a FPU subfunction. The low level properties of an IP are listed as features. For example, precision is a feature of a FPU IP. An attribute is a value of a feature. For example, single or double are values of a feature, precision. At the lowest level, operations describe the composition of functions and subfunctions. For example, "exponent difference" operation for denormalization subfunction.

The expert further refines the PBM using combination of following notations 1) library of template code fragments with corresponding annotations to infer behavioral functions, 2) expected output values of a behavioral function when a reference input is applied, and 3) design assertions.

The library of template code fragments are selected by design experts as a pattern for concept to code mapping. The functions and operation ontological concepts correspond to code fragments (sequence of control and data operations) in RTL and get associated with a set of rules in KB. The concepts inferred by pattern matching rules are further confirmed through application of test vectors. The KB also has formal properties for a design as a library of assertions which are created in property specification language [18].

We explain the property based model using an example for synchronous FIFO.

1) Example - Property based model for FIFO: The property based model of a FIFO is represented in Figure 4. The model has a hypothesis about FIFO design composition in terms of behavioral functions. In Figure 4 "push data", "pop data" and "exception handling" are categorized as behavioral functions for FIFO design. The expert knowledge about a FIFO design is also captured in the model. For example, FIFO is categorized as a data buffer and as a single clock-domain design. The designer also writes assertions to capture FIFO design behavior. The template code fragments associated with the model provide hints for existence of behavioral functions. For example, Table II lists FIFO RTL code fragments and their corresponding annotations.

C. RTL rule-base

The RTL-KB is in form of CLIPS fact representation. The properties of the CLIPS code are made into CLIPS rules with the assistance of a code analysis tool. The tool allows an expert user to provide an annotation for a fragment of RTL code. The user identifies the annotation and associates it with sample CLIPS facts from the converted RTL code. The tool then assembles the code fragment as antecedent and annotation as consequent into a set of CLIPS rules. The resulting rules then automatically annotate sections of trusted RTL code and low-level annotations are used for performing higher-level annotations. The expert's skill in recognizing code fragments becomes part of the system and similar code fragments in untrusted RTL get annotated with an equivalent concept. The tool simplifies the work flow for the user who need not know the form and syntax of CLIPS rules.

Our methodology has a confidence factor (CF) associated with each rule in RTL rule-base to support partial matching and accommodate expert knowledge about significance of a design property. The CF given to the rule by an expert is in the range of 0 to 1. CF is the likelihood of being accurate in making a inference when a particular design property exists in RTL code. The CF allows the rule base to identify high level design functions even for partial matching of the design

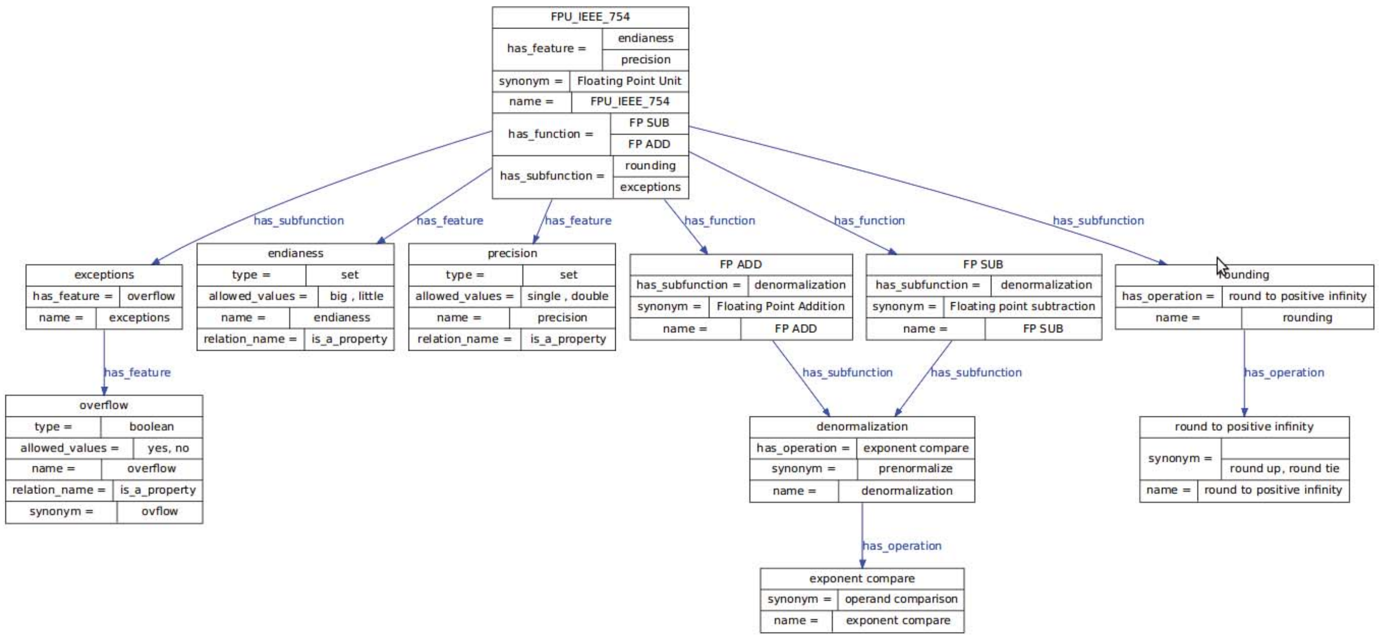

FPU_IEEE_754
has_feature = endianess, precision
synonym = Floating Point Unit
name = FPU_IEEE_754
has_function = FP SUB, FP ADD
has_subfunction = rounding, exceptions
has_subfunction
has_feature
has_feature
has_function
has_function
has_subfunction
exceptions
has_feature = overflow
name = exceptions
endianess
type = set
allowed_values = big , little
name = endianess
relation_name = is_a_property
precision
type = set
allowed_values = single , double
name = precision
relation_name = is_a_property
FP ADD
has_subfunction = denormalization
synonym = Floating Point Addition
name = FP ADD
FP SUB
has_subfunction = denormalization
synonym = Floating point subtraction
name = FP SUB
rounding
has_operation = round to positive infinity
name = rounding
has_feature
overflow
type = boolean
allowed_values = yes, no
name = overflow
relation_name = is_a_property
synonym = ovflow
has_subfunction
has_subfunction
has_operation
denormalization
has_operation = exponent compare
synonym = prenormalize
name = denormalization
round to positive infinity
synonym = round up, round tie
name = round to positive infinity
has_operation
exponent compare
synonym = operand comparison
name = exponent compare

Fig. 3: Design ontology for FPU design domain

TABLE II: Example - FIFO RTL code fragments with annotations performed by rule-base

S.No	Code fragment	Annotation(first level)	Annotation (second level)
1	always @(posedge pclk or negedge presetn) if(!presetn)	pclk annotated as clock and resetn annotated as active low reset.	The always block gets annotated as "clocked process".
2	rd_addr <= rd_addr + 1 wr_addr <= wr_addr + 1	rd_addr and wr_addr annotated as up counters	
3	data_out <= memory[rd_addr]	rd_addr is annotated as read address	Based on first level annotations in (2) and (3) rd_addr is annotated as read pointer. The corresponding code blocks are annotated as pop logic
4	memory[wr_addr] <= data_in	wr_addr is annotated as write address	wr_addr is annotated as write pointer. The corresponding code blocks are annotated as push logic.
5	assign incr_fifo_word_cnt = (push,pop = 2'b01)? 1'b1 : 1'b0	push and pop annotated as control signals	

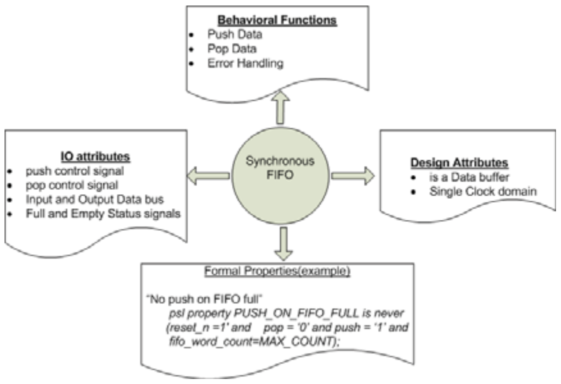

Behavioral Functions
Push Data
Pop Data
Error Handling
IO attributes
push control signal
pop control signal
Input and Output Data bus
Full and Empty Status signals
Synchronous FIFO
Design Attributes
is a Data buffer
Single Clock domain
Formal Properties(example)
"No push on FIFO full"
psl property PUSH_ON_FIFO_FULL is never (reset_n ='1' and pop = '0' and push = '1' and fifo_word_count=MAX_COUNT);

Fig. 4: Property based model of a FIFO design

properties. The CF's are merged for rules inferring a function at same hierarchical level using parallel combination formula. We produce the CF of inferring a higher level module, based on the existence of lower modules, by the average (weighted) of the CF's of lower level modules. Following is a description of various category of rules.

a) Behavioral analysis rules: These rules identify in an untrusted RTL, the existence of function, operations and features specified in the PBM. For example, following is a RTL code fragment, which implements count leading zeros (LZ) subfunction. The associated pseudo rule which gets implemented in CLIPS to infer a LZ function is also listed.

```
"RTL code fragment 1 for identifying LZ function"
elsif (fracAddOut_s3(31 downto 28) = "0001") then
        tmpOUT := "00010";
elsif (fracAddOut_s3(31 downto 27) = "00001") then
        tmpOUT := "00011";

"Pseudo-rule: infer leading zeros function"
1) A block of code in a combinational process which has
multiple condition statements or loop construct with
common condition variable, (say x).
2) x is compared against a constant value.If the condition
is true, a variable (say y) gets updated.
3) When control falls from one condition statement to
another the value of y keep on incrementing by 1.
```

The KB system keeps on improving as more designs are analysed and rules get added to capture variations in RTL implementation. For example, If KB contains FPU designs with different RTL implementation of LZ function, then each of those different code fragments can be used as a pattern for a new rule to infer LZ function. The addition of trusted designs and new design domains to the KB increases the accuracy of rule based inference process. Following is CLIPS format of the LZ pseudo rule

```
(defrule FPU_BEHAV_10::LEADING-ZERO-ANNOTATE
(declare (auto-focus TRUE))
(stmt (cat BLOCK)(id ?f1pid) (stmts ?f1id ?f2id ?f3id))
(stmt (parentid ?f1pid) (id ?f1id)(cat RANGE) (text ?f1txt&:
 (member$ ?f1txt(create$ TO DOWNTO)))(ins ?sigID ?rangeID1&:(stringp
 ?rangeID1) ?rangeID2&~?rangeID1&:(stringp ?rangeID2))(outs ?f1out))
(stmt(parentid ?f1pid) (id ?f2id) (cat MATH) (text ==) (ins ?f1out
 ?f2in2&:(stringp ?f2in2))(outs ?f2out))
(stmt (parentid ?f1pid) (id ?f3id) (cat SEQ) (text IF) (ins ?f2out)
      (stmts ?f3stmt1 ?f3stmt2))
(stmt (parentid ?f3id) (id ?f3stmt1) (cat BLOCK)(stmts ?f4stmt1))
(stmt (parentid ?f3stmt1) (id ?f4stmt1) (cat MATH) (text ASSIGN)
      (ins ?f3In) (outs ?f3it ))
(test (= (+(-(string-to-field ?rangeID1)(string-to-field ?rangeID2))
      1)(str-index "1" ?f2in2)))
=>
(bind ?process (find-process ?f1pid))
(assert (inference (module ?module)(process ?process)
        (annotation "LZ")(cf 0.9)))
```

b) Datapath rules: These rules use design datapath knowledge to find associations of an unannotated process with annotated processes. The inference done by these rules have low CF. In our approach, we consider concurrent assignment statements also as processes. For a "FPU Add" function in the PBM, the expert specifies its datapath as "fraction addition" subfunction followed by "LZ" subfunction and then "fraction left-shift" subfunction. In Figure 5, which shows a fragment of a process dependency graph (pdg) for FPU, process P0 and P1 are annotated with "fraction add" and "fraction left shift" subfunctions respectively. The process P1 remains unannotated as antecedent code pattern for LZ rule does not match with the RTL. In that case, "FPU add datapath" rule annotates P1 with LZ subfunction.

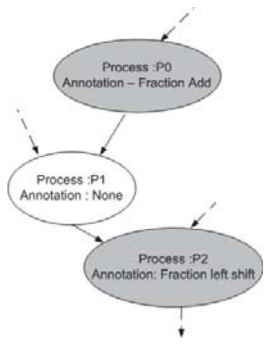


Fig. 5: Fragment of pdg for floating point add function

c) Syntactic rules: This rule-set analyzes RTL code structure and infers common design concepts like counters, look up table, memory components, FIFO, FSM etc. The processes get annotated as sequential, combinational, control-only process, and for occurrence of arithmetic operations. These rules also perform custom RTL lint checks. Following is an example of syntactic lint rule.

```
Pseudo Rule :
a) Generics should have default value assigned in entity declaration.
b) Input signals to the design should be registered.
c) Output signals from the design should be registered outputs.
```

d) Evaluation Rules: In our approach, the RTL is parsed and represented as CLIPS fact. The CLIPS program is then run to simulate the execution of the VHDL code and it acts as an event based simulator. Every behavioral function is associated with a evaluation rule, which runs a sample test vector associated with a behavioral function. The success of an evaluation rule increases the CF of the function inferred by behavioral analysis rule.

e) I/O port rules: I/O port signals describe the interface for the design. In our approach, an expert associates a label with an IO signal name in trusted RTL to map names from different vendors to an equivalent concept. For example, "div_zero_o", "div_by_zero" signal names are labelled as "divide by zero" for FPU domain. This is captured in an I/O label dictionary. We then assume that the I/O signals of the untrusted IP are also labeled by users using concepts from the IO label dictionary. This aids behavioral analysis rules to annotate RTL code fragments. For example, if input port "rnd" is labelled as rounding signal, then RTL code fragments where "rnd" is a control signal are annotated as rounding subfunction.

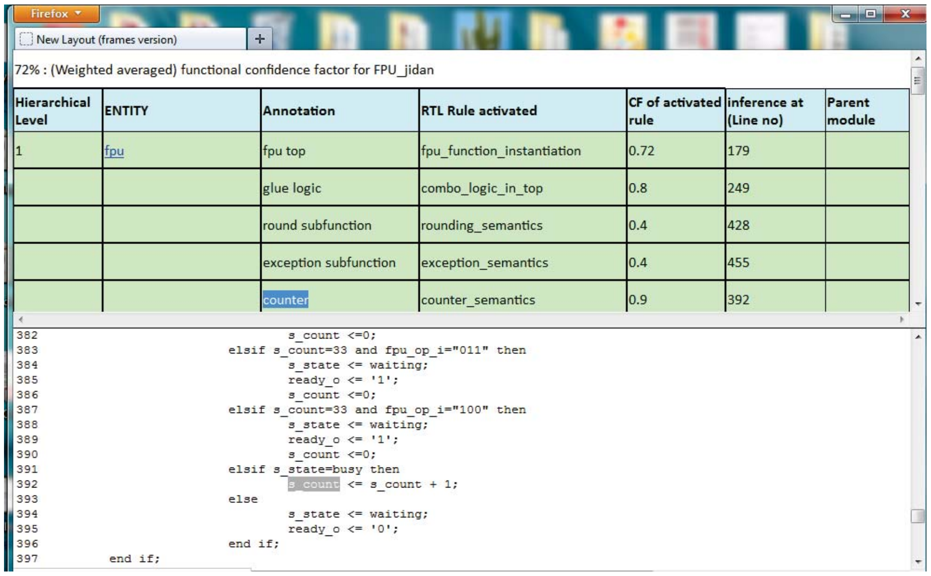
IV. Results

Coverage is an important concept in RTL design verification and is defined in terms of code/functional coverage metrics. The coverage metrics in our case concern functionality identification coverage, thus our metrics are expressed in terms of confidence factors. We identify the functionality of soft IP design modules based on the existence of their behavioral properties. This process does not necessarily guarantee valid design behavior. Therefore we cannot use traditional coverage metrics such as code coverage. For quantifying our results, we use functional identification (Id) coverage as a metric, which is percentage of functions listed in PBM model that can be inferred from an untrusted RTL.

We have tested our approach by developing a prototype KB system using CLIPS. The tool supports VHDL and for IPs in verilog, we have used opensource verilog to vhdl converter [19]. The KB was then applied to perform RTL analysis of FPU IPs downloaded from opencores.org. The FPU PBM has 11 functions, 15 subfunctions and 89 features. In Table III, the Id coverage for each IP is proportional to number of high level design functions inferred. If a function gets inferred but all of its subfunctions and properties cannot be inferred then CF for a function is low and it decreases overall CF. The CF listed for each IP is the average confidence of the rule-base about the inferences performed for the IP. Figure 6 provides an example of html based annotation report that was generated for FPU. If the Id coverage for a 3rd Party IP is low, then it is a hint to the user about possible design quality issues. The user can review the annotation reports to find the design properties listed in PBM but missing in the 3rd Party IP.

TABLE III: RTL analysis results using RTL rule base

Design information					**Analysis results**			
IP name	**No. of design modules**	**No. of lines of code**	**input ports**	**output ports**	**No. of rules fired**	**Properties identified**	**Identification-coverage**	**Overall Confidence factor**
fpuOpencores1 [20]	11	1176	5	9	69	34	70%	72 %
fpuOpencores2 [21]	14	2396	5	11	82	42	77%	80 %
fpuOpencores3 [22]	24	3259	4	1	31	15	55%	56 %
fpuOpencores4 [23]	7	1912	7	7	46	22	68%	63 %



72% : (Weighted averaged) functional confidence factor for FPU_jidan

Hierarchical Level	ENTITY	Annotation	RTL Rule activated	CF of activated rule	inference at (Line no)	Parent module
1	fpu	fpu top	fpu_function_instantiation	0.72	179	
		glue logic	combo_logic_in_top	0.8	249	
		round subfunction	rounding_semantics	0.4	428	
		exception subfunction	exception_semantics	0.4	455	
		counter	counter_semantics	0.9	392	

```
382                         s_count <=0;
383                 elsif s_count=33 and fpu_op_i="011" then
384                         s_state <= waiting;
385                         ready_o <= '1';
386                         s_count <=0;
387                 elsif s_count=33 and fpu_op_i="100" then
388                         s_state <= waiting;
389                         ready_o <= '1';
390                         s_count <=0;
391                 elsif s_state=busy then
392                         s_count <= s_count + 1;
393                 else
394                         s_state <= waiting;
395                         ready_o <= '0';
396                 end if;
397         end if;
```

Fig. 6: Html annotation report

V. Conclusion

We have presented a knowledge-guided methodology to perform RTL analysis of third-party Soft-IPs. Our validation approach can assist design engineers in understanding of RTL behavior. The confidence level and identification coverage results can be used as metrics for IP analysis. The key challenge for the rule-base is to use successful combination of behavioral analysis and evaluation rules to make an inference. For future work, we plan to use assertion synthesis to generate assertions from RTL and then perform a rule-based analysis of assertions in CLIPS.

References

[1] W. K. Lam, *Hardware Design Verification: Simulation and Formal Method-Based Approaches*, Ist edition, Prentice Hall, 2006.

[2] F. Wickberg, *HDL code analysis for ASIC in Mobile Systems*, *liu.diva-portal.org/smash/get/diva2:24142/FULLTEXT01*

[3] W. Wogerer, *A Survey of Static Program Analysis Techniques*

[4] D. Esposito, *Static Code Analysis and Code Contracts*, MSDN Magazine, August 2011

[5] M. C. Hansen, H. Yalcin, J. P. Hayes, *Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering*, IEEE D&T of Computers, July-September, 1999.

[6] W. Li, Z. Wasson, S. A. Seshia, *Reverse Engineering Circuits Using Behavioral Pattern Mining*, IEEE Int. Symposium on Hardware-Oriented Security and Trust (HOST) 2012.

[7] K. Werner, *Can IP quality be objectively measured?*, DATE 2004.

[8] Z. Meng, G. Minglun, P.C.H Chan, *A practical IP quality measurement framework*, ASIC, 2007. ASICON'07.

[9] Chaouat, L.; Vachoux, A.; Mlynek, D., *A method to Implement a Knowledge-Based System for Fast prototyping of hardware designs*, Int. Conf. On Industrial Engineering and Applications. 1996.

[10] Lopez, M.L.; Iglesias, C.A.; Lopez, J.C., *A knowledge-based system for hardware-software partitioning*, DATE 1998.

[11] Y. Naveh; M. Rimon; I. Jaeger; et al., *Constraint-Based Random Stimuli Generation for Hardware Verification*, AI Magazine, Vol 28, 2007

[12] J.C. Giarratano, G.D. Riley, *Expert Systems: Principles and Programming*, 4th ed.: Thomson Course Technology, 2005.

[13] G. Riley, *http://clipsrules.sourceforge.net/*

[14] IEEE, *IEEE 754-2008 Standard for Floating point arithmetic*.

[15] Zhanjun Li, Karthik Ramani, *Ontology-based Design Information Extraction and Retrieval*, Journal of Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 2007.

[16] Nirenburg, S., Raskin, V., *Ontological Semantics*, MIT Press 2004.

[17] *http://protege.stanford.edu/*.

[18] IEEE, *IEEE P1850 - Standard for Property Specification Language (PSL)*

[19] K.L. Ghosh, *http://www.edautils.com/verilog2vhdl.html*

[20] R. Usselmann, *http://opencores.org/project,fpu*.

[21] A. Jidan, *http://opencores.org/project,fpu100*.

[22] M. Guillermo, *http://opencores.org/project,fpuvhdl*

[23] D. Lundgren, *http://opencores.org/project,fpu_double*