Cross-correlation of Specification and RTL for Soft IP Analysis

Bhanu Singh[1], Arunprasath Shankar[1], Francis Wolff[1], Christos Papachristou[1], Daniel Weyer[2], and Steve Clay[2]

[1]Dept. of EECS, Case Western Reserve University, Cleveland, USA
[2]Rockwell Automation, Cleveland, USA

***Abstract*—Semiconductor companies often use 3rd party IPs in order to improve their design productivity. In practice, there are risks involved in using a 3rd party IP as bugs may creep in due to versioning issues, poor documentation, and mismatches between specification and RTL. As a result of this, 3rd party IP specification and RTL must be carefully evaluated. Our methodology addresses this issue, which cross-correlates specification and RTL to discover these discrepancies. The key innovative ideas in our approach are to use prior and trusted experience about designs, which include their specs and RTL code. Also, we have captured this trusted experience into two knowledge bases (KB), Spec-KB and RTL-KB. Finally, knowledge base rules are used to cross-correlate the RTL blocks to the specs. We have tested our approach by analyzing several 3rd party IPs. We have defined metrics for specification coverage and RTL identification coverage to quantify our results.**

I. Introduction

IP reuse is a common design approach to develop system on chip (SoC) designs. SoC design teams often source standard based 3rd party IPs. In an ideal reuse scenario, 3rd party IPs can be seamlessly integrated into the SoC [1]. In practice, however there are risks involved in using 3rd party IPs. Design quality issues in a 3rd party IP, if discovered later than the RTL sign-off stage, increase the turnaround time and in the worst case may lead to chip re-spins. Therefore, to mitigate the risk of 3rd party IP reuse, the IP specs and RTL must be carefully evaluated to discover any discrepancies early in design cycle.

We consider a common scenario, where a SoC design team develops a IP specification (trusted spec) document but outsources the IP design and verification to a 3rd party design house. The SoC team finally receives the IP block as soft IP core, which has a synthesizable RTL code description and a detailed functional spec document. An important issue is that the RTL design corresponds without discrepancies to its original trusted spec. Our method addresses this issue.

In current practice, spec analysis is primarily done manually by reading a vendors functional spec and comparing it against the original trusted spec. There is a growing concern that this manual process is slow and error prone. The state-of-the-art in RTL validation can be broadly classified into (a) Simulation, and (b) Formal methods based on either theorem proving or model checking. Formal approaches may be challenged by the state explosion problem in attempting to formally verify large designs [2]. Simulations are very expensive as they require long run times operating on large functional vector sets. Thus, employing formal and functional testing may not provide high confidence level at a reasonable cost.

978-3-9815370-2-4/DATE14/© 2014 EDAA

In this paper, we propose a knowledge-guided methodology to analyze a 3rd party soft IP, possibly untrusted, provided by its RTL code description and a trusted specification document. Our approach uses prior and trusted experience about designs, including their specs and RTL code, and captures this experience in two KB, Spec-KB and RTL-KB. Knowledge Base rules are then used to find correspondences of RTL with its original trusted spec and infer its relation block by block to the spec document structure. The design behavior inferred from RTL is back annotated to corresponding spec sections. In our approach, exploitation of prior and trusted design experience through both Spec and RTL KBs is a key benefit in contrast to using formal methods and functional simulations.

A. Related Works

The individual elements of our technique can be roughly related to work in following areas: (a) Specification analysis, (b) Reverse engineering, and (c) IP quality evaluation.

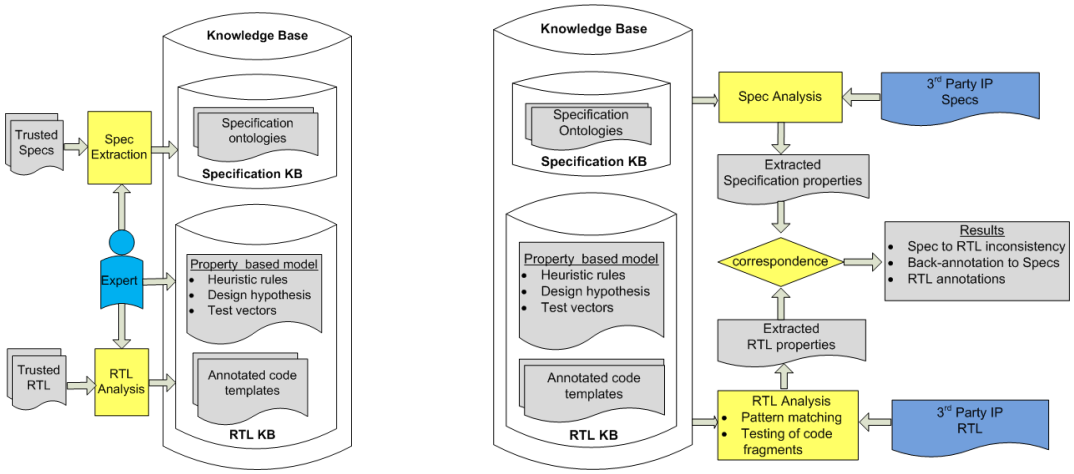
Natural language processing (NLP) based techniques, have been proposed for extracting formal specifications from an informal spec document [3] [4]. The in-depth NLP based approach is very complex and computationally expensive. NLP is not required for our approach as we focus on limited domain specific aspect of text and do not process documents for general linguistic context.

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

Various approaches have been proposed for IP quality evaluation [7] [8]. Quality metrics have been proposed such as Quality Intellectual Property (QIP) by virtual socket interface alliance (VSIA) [7]. Our approach provides additional specification and RTL identification coverage metrics to quantify IP design quality. In industry, the so called lint tools are primarily used to check RTL design quality [9] but these tools do not analyze specification to RTL correspondences.

II. Methodology Overview

The main elements of our technique are: (a) a KB of trusted Specs and RTL, (b) property based models for common 3rd party IP blocks (c) Specification analysis, (d) RTL analysis,



(a) KB generation (b) KB application for third party IP analysis

Fig. 1. KB system to determine Spec and RTL correspondence

and (e) back-annotation of inferred RTL functionality to specification. Fig.1 provides an overview of our approach. We have used the expert system shell CLIPS, with its powerful inference engine to help us generate the Spec and RTL KBs and perform rule based analysis [10]. The generation of KB is a ongoing process, as KB is enhanced with each addition of new IPs. Every IP in the KB is considered as trusted design, built previously by experienced designers. The 3rd party IP analysis involves a rule-based search process, which checks the existence of trusted design properties in an untrusted design. Thus correspondences between 3rd party RTL and its original trusted spec are inferred through the KB search by matching CLIPS rules.

Our method can be applied to standard based soft IPs. In this context, standards are trusted public documents that establish specifications that are universally understood to ensure compatible functionality and interoperability. A standard acts like a high-level specification and defines key concepts (functions and properties) for a design. Soft IPs from different vendors that are compliant to a standard have concrete behavioral correspondences. For example, every IEEE-754 compliant floating point unit (FPU) design should support the following rounding modes: “round to zero”, “round up”, “round down” and “round to nearest even” [11].

We also use the concept of design domain, which we define as a family of Soft-IP designs that have common application, such as a bus interface domain. We generated KB from libraries of standard-based designs and organized it as per various design domains. In the following sections, we describe each element of our technique.

III. Knowledge base Generation

Our method employs Spec-KB and RTL-KB, which are described below

A. Specification KB

The Spec-KB consists of following components:

1) specification ontology: A spec document is generally used to specify design behavior. It uses variety of notations that include text, diagrams, and tables. We define text-objects as words or phrases used to specify design properties. For example, “synchronous” and “asynchronous” are text-objects. Through analysis of different vendor specs, we observed that for a particular design domain, specs exhibit commonality of text-objects. These text-objects are identified from a corpus of specs and then organized into a conceptual network, which we term as “spec ontology”. An ontology is defined as a knowledge representation model to explicitly represent a domain by defining its concepts and their relationships [12].

A spec ontology consists of objects, features and their attributes organized in a hierarchy using relationships. Fig.2 provides an overview of the ontology fragment for the ARM AXI Bus. Each high level concept in the ontology 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 a specification model for the design. The ontologies that we have developed describe the design using four-level-deep functional decomposition. The first level is soft IP design domain along with its listed features. The second level has function objects. The third level consists of subfunction objects, which are further decomposed in terms of leaf level operations. For example, in Fig.2 data-transfer is categorized as an AXI function and “flow-control” is its subfunction. We have developed tools to assist experts in ontology creation. We first filter common words such as pronouns, conjunctions etc. by using a custom stop word dictionary. We then perform spec word frequency analysis and location analysis on a corpus of specs to provide input to an expert about lists of important words for a design domain.

We have also developed a “Spec-Extractor” tool to automate the extraction of text-objects from a third-party IP spec document. A spec document is analyzed by first converting it from PDF into XML format using Adobe tools. The spec-

OBJECT	FEATURE	SYNONYM	RELATION	TYPE	ATTRIBUTE
axi4s	axi4 stream protocol		is-a-property	bool	yes, no
axi4s	data transfer	data exchange	is-a-function	bool	yes, no
axi4s	interconnect		is-a-function	bool	yes, no
axi4s	signaling	AXI ports, AXI4S I/O	is-a-signal-list	bool	yes, no
…	…	…	…	…	…
signaling	tvalid	valid transfer	is-a-signal	bool	yes, no
signaling	tready	master/slave ready	is-a-signal	bool	yes, no
signaling	tstrb	byte qualifier data	is-a-signal	bool	yes, no
…	…	…	…	..	..
data transfer	slave		is-a-property	bool	yes, no
data transfer	master		is-a-property	bool	yes, no
data transfer	side band signaling		is-a-property	bool	yes, no
data transfer	flow control		is-a-subfunction	bool	yes, no
…	…	…	…	…	…
interconnect	arbitration		is-a-subfunction	set	first come first serve, fixed priority

Fig. 2. Fragment of specification ontology for AXI bus

extractor then parses the XML and generates a hierarchical tree based on the location of each spec word in a particular section, subsection, paragraph and sentence of the document. Since each word is a leaf node in the tree, the key idea is the path back to the root. A unique location vector is generated for every word derived from corresponding XML tags for document structure. Spec words are further clustered under categories as, (i) Spec ontology words, (ii) RTL symbols (port and module names) (iii) acronym/abbreviations/numbers, (iv) custom stop words, (v) English dictionary words, and (vi) unknowns. This captured data is outputted as CLIPS facts, which then gets analyzed by the spec rule base (see below). Each trusted spec gets represented in a machine readable format by populating a uniform CLIPS fact template with text-objects existing in that particular spec. This representation enables us to compare the spec of a new design with existing designs in the KB. The Spec ontology helps in extraction of salient features of a design from textual spec documents.

2) Specification Rule-base: The rule-base analyzes CLIPS fact based representation of a 3rd party IP spec by finding correspondences with the spec KB. The rule-base has multiple level of inference rules. The higher level rules perform inferences by combining lower level inferences. Following is description of various categories of spec rules.

a) Location rules: These rules use location proximity information of text-objects in the spec to associate design features with their attributes or find negative phrases associated with features. For example, for FPU designs, precision is a feature and it has attribute value of "single" or "double".

b) Function Rules: These rules infer design functions based on occurrence of text-objects that identify low level operations and sub-functions listed in the spec ontology. We may have a partial match of features in a spec with certain features that are more important as they provide hints of the existence of high-level functions. For example, the existence of a lookup table or memory component for FPU designs can be used as hint to infer the occurrence of a floating-point divider. The I/O names for bus protocol designs can also be used as hints to infer the associated bus interface function. For example, if a spec has partial matching with signal objects listed in Fig. 2, then it is a hint for the existence of AXI bus. The rules also check if any essential design property key to making inferences about a design function is missing in a third-party IP spec, such as overflow property of FPU design.

c) IO property based: These rules check I/O inconsistency between spec and RTL by testing if port names and their attributes specified in a spec document match with RTL description. The I/O tables listed in specs are parsed from XML version of specs to infer I/O connectivity specified for the design. The IO port information for each RTL entity/module is extracted from the RTL port definitions.

B. RTL KB

The RTL-KB consists of following components:

1) RTL-CLIPS: This format is a semantically equivalent translation of RTL code into CLIPS facts. It helps the rule-base to semantically and syntactically infer RTL properties. We have developed tools to perform RTL to CLIPS translation and a simulation environment for RTL-CLIPS.

Table I gives some examples of VHDL language elements and their CLIPS templates. We define two types of fact templates and instances of these templates as facts represent RTL in CLIPS. The var template captures attributes associated with a signal/variable declaration. The $stmt$ template captures attributes of a control/data operation. For example, inputs, output and type of operator used. 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.

TABLE I
EXAMPLES OF HDL REPRESENTED IN KNOWLEDGE BASE

VHDL Element	CLIPS Template
signal clk : std_logic; ...	(var (parentid gen3) (id gen6) (text clk)(class signal))
process (clk) ...	(stmt (parentid gen13) (id gen17)(cat BLOCK CONC-SEQ)(ins gen6) (stmts gen19 gen20))
elsif rising_edge(clk) then	(stmt (parentid gen23) (id gen26) (cat SEQ IF) (ins gen6) (stmts gen27 gen29))
cnt <= cnt + 1; ...	(stmt (parentid gen26) (id gen27) (cat MATH ADD) (ins gen8 "1") (outs gen8))

$Stmt$ 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 ADD" for addition. The statements in a code block make 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. Within a process, flow of control constructs include sequential, conditional, loop and function. Sequential control is defined by an ordered list of ids in the $stmts$ multislot of the parent $stmt$. These ids appear in the id slot of each child. Conditional and loop control are defined as a parent $stmt$ with command category "SEQ IF" or "SEQ WHILE" and a conditional variable reference in the ins multislot and one or two ids in the $stmts$ multislot. Function calls are defined as a $stmt$ with command category "SEQ FUNC $name$" for the function name and ins and $outs$ multislots with references to arguments and results.

2) A Property Based Model (PBM) for the design: A property is a partial specification which contains some element of expected design behavior. In our approach, the behavior of a

standard based design is represented by collection of its partial specifications. The properties of trusted designs are captured in the KB through a PBM. The expert generates this model using combination of following notations (a) hypothesis about design behavior, (b) library of template code fragments, and (c) test vectors for expected input/output behavior.

For an IP, its hierarchical functional decomposition results in behavioral functions, which at the highest abstraction level, are captured through the spec ontology. The PBM for a design is further refined by experts providing template code fragments, which act as pattern for concept to code mapping. The functions and operations in spec ontology correspond to a set of RTL code fragments (sequence of control/data operations), captured through rules in RTL-KB. For example, Table II lists some template code fragments and their annotations (spec-words listed in corresponding domain spec ontology).

The properties of the CLIPS code are semi-automatically made into CLIPS rules with the assistance of a code analysis tool. The tool allows an expert 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 concepts inferred by pattern matching rules are further confirmed through application of test vectors. We explain the PBM using an example for synchronous FIFO, shown in Fig. 3.

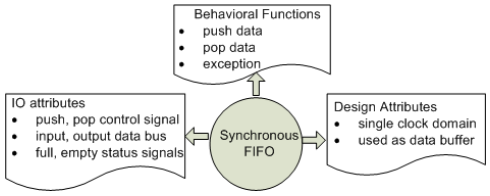


Fig. 3. Property based model of a FIFO design

Example -: The model has a hypothesis about FIFO design composition in terms of its behavioral functions: "push data", "pop data", and "exceptions". The expert knowledge about a FIFO design is also captured. For example, synchronous FIFO is categorized as a single clock-domain design. The template code fragments associated with the model as shown in Table II provide hints for existence of these behavioral functions.

3) RTL Rule-base: The RTL-rulebase captures heuristic knowledge of design experts. The KB keeps on improving as more designs are analyzed and more rules get added to capture variations in RTL implementation. This increases the accuracy of the rule based inference process. Following is a description of various rule categories.

TABLE II
EXAMPLE - TEMPLATE RTL CODE FRAGMENTS AND THEIR ANNOTATIONS

Code fragment	**Annotation (first level)**	**Annotation (second level)**
process (pclk or presetn) if presetn = '0' then rdAdr <= "0"; elsif rising_edge(pclk) then	pclk and presetn annotated as clock and active low reset.	process block gets annotated as "clocked process".
rdAddr <= rdAdr + 1 wrAdr <= wrAdr + 1	"up-counter" annotation	
dout<= mem(conv_integer(rdAdr));	rdAdr annotated as "read address"	further annotated as read pointer

a) Syntactic rules: This rule-set infers common design concepts like counters, memory components, FIFO, FSM etc. A "process" in VHDL get annotated as sequential, combinational, control-only, and for existence of arithmetic operations.

b) Behavioral analysis rules: These rules identify properties of an untrusted RTL by finding correspondences with RTL KB. The inferences done by rules at lower level are used to perform higher level inference. Each rule uses a combination of template code fragments as a pattern and on successful firing provides an annotation. Following is an example of a rule which infers "logical left-shift" operation.

```
(defrule MAIN::infer-LSL
(stmt(parentid gen0)(cat DESIGN)(text ?module))
(stmt(cat CONST)(text ?x&~:(str-index 1 ?x)) (outs ?y))
(stmt(parentid ?p)(cat MATH CONCAT) (ins ?f2in1 ?y)(outs ?out1))
(stmt(parentid ?p)(cat MATH WHEN)(ins ?cmpResult1 ?out1 ?f3in2))
(stmt(parentid ?p)(cat MATH ==)(ins ?sel ?const1)(outs ?cmpResult1))
(stmt(cat CONST)(text ?constVal1&:(eq(string-to-field ?constVal1)
  (str-length (proper-str ?x)))) (outs ?const1))
=>
(assert(inference(module ?module)(function LSL)(cf 0.9))))
```

c) Register Meta-data rules: We define a register description template as shown in Table III and it uses meta data tags for register specification. Each register in spec is specified using meta-data tags to capture its name, address, reset value and access type. The register meta-data rules identify mismatches between register specification in Spec document and register implementation in RTL. For example, inconsistency in register label/address etc.

TABLE III
PROPOSED REGISTER DESCRIPTION TEMPLATE

module name	**base address**	**bus protocol**	**version**	**date**	**author**	
fpuReg	0x400	AHB	2.1	xyz	abc	
register name	**address offset**	**field name**	**reset value**	**index**	**access type**	**comments**
OP_SEL	0x0					operation select
		RSVD	0x0	31:2	RO	
		OPCODE	0x0	1:0	RW	2'b00 = Add; ...

d) I/O port rules: For each design domain, an expert associates a label with signal names to map names from different vendors to an equivalent concept. For example, for FIFO designs "push" and "write" signals are labelled as "fifo write". This is captured in an I/O label dictionary. For a new design, automatic port name resolution is performed using stemming and lemmatization techniques [13]. This associates each port name with an equivalent concept from the I/O label dictionary. The generated I/O label file is reviewed by a user and is provided as an input to the rule-base.

e) Datapath rules: These rules use design composition knowledge provided by experts in PBM. These rules annotate unidentified code blocks using annotation information of neighboring code blocks. For example, for a "FPU Add" function in the PBM, its datapath has "fraction addition (FrAdd)" subfunction followed by "leading zeros (LZ)" subfunction and then "fraction left-shift (FrLS)" subfunction [15]. If two code-fragments in RTL are annotated as "FrAdd" and "FrLS", then the data dependency is analyzed to annotate a associated code fragment as "LZ" subfunction.

f) Evaluation Rules: In our approach, a CLIPS program is run to simulate the execution of the VHDL code. The KB has reference test-vectors associated with each behavioral function and these are developed as part of trusted designs in the KB. The behavioral analysis and data path rules first identify entities in an untrusted RTL that have matchings with functions in PBM. The IO Port rules further provide concept labels to RTL signals of 3rd party IP. The rule-base then generates input test values for an entity in untrusted RTL using a test vector in KB and IO concept labels. The user reviews the input values before they are applied as evaluation rules to simulate 3rd party IP.

4) Confidence levels: Our methodology has a confidence factor (CF) associated with each rule to support partial matching and accommodate expert knowledge about significance of a design property. CF given to a rule by an expert is in the range of 0 to 1. CF is the likelihood of being accurate in inferring a design functionality when a particular design property exists in Specs or RTL code. CF allows the rule base to identify high level design functions even for partial matching of properties. CFs are merged for rules inferring a function at the same hierarchical level using a parallel combination formula. Fig. 4 provides an example of CFs merging. We produce the CF of inferring a higher level module, based on the existence of lower modules, by the weighted average CFs of lower level modules.

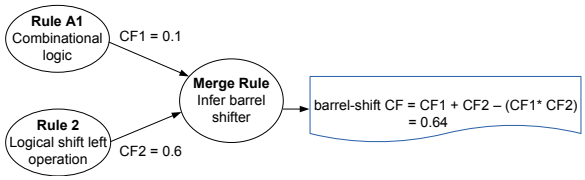


Fig. 4. Confidence factor merging using parallel combination formula

IV. Back Annotation technique and Metrics

A. Back annotation Technique

The objective of back annotation is to map RTL modules to the corresponding sections in spec. We assume that the spec has IO port list specified for each RTL entity. The spec analysis method uses location vectors of text-objects to annotate the corresponding spec sections/subsections. The RTL modules get annotated by the RTL analysis method. A particular spec section to RTL entity/module correspondence is established through the existence of module name, matching annotations and IO port list in that section. For example, in Fig. 5 #1, #2, #3 refer to back-annotation of RTL entity/ports to sections in spec. Finally, design properties annotated to the section are matched against RTL annotations for the entity. CF is assigned for both complete matches and partial matches of the annotations. The back-annotation follows a bottom up procedure where CFs generated for matching lower level modules are summed and averaged progressively to find the CF for a higher level module. Fig. 5 provides an overview of back annotation process

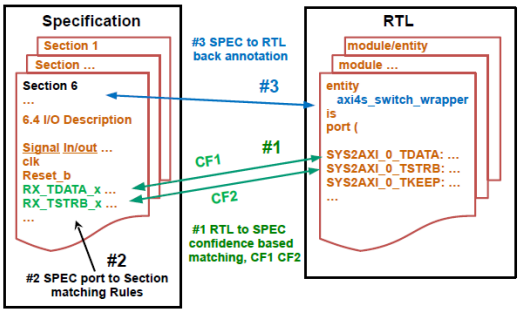


Fig. 5. Back Annotation overview

B. Metrics

1) Spec coverage (SC): A Spec section is considered **covered** if a function and its properties listed in spec ontology are annotated to that section. Since functions are first level objects, inference of a high-level function by spec analysis and its location in "section" text covers all lower level subsections. If a high-level function could not be directly identified but its properties and sub-functions are found in a subsection, then the corresponding section is annotated with the function with a confidence level (CF) calculated by the rule-base. If the CF is below a user-defined threshold, the section is considered uncovered. SC results provide feedback about sections that have no correspondence with spec-KB and RTL. These sections need to be carefully reviewed by the user.

2) Identification (Id) Coverage: Id coverage is defined as the percentage of functions listed in PBM model that can be inferred from an untrusted RTL. 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 value for an IP is the average confidence of the rule-base about the inferences performed for the IP.

V. Summary of Results

We have developed a prototype KB system based on powerful CLIPS inference engine for the following design domains: FPU, ARM2 and timer. With our industrial collaborators, we developed trusted designs (Specs & RTL), spec ontologies and Spec & RTL rule-bases for these design domains. The FPU, ARM2 and Timer spec ontology contained 115, 134 and 39 features respectively. We obtained vendor FPU IPs, AMBER ARM2 IP and Timer IP from opencores.org. We considered each vendor IP as untrusted design. We performed analysis of spec and RTL for each untrusted IP.

TABLE IV
SPEC ANALYSIS RESULTS

Spec Document Information				Spec Extractor Output						Spec Rule-base Analysis Output			
IP Name	No. of Pages	No. of Sections	Domain	Total words	Ontology matches	Acronym/ abbreviat.	Number/ symbol	English Dictionary	un-known	Functions inferred	Features inferred	Sections annotated	Spec coverage
FPU_IP1 [14]	6	4	FPU	710	51	24	26	563	46	6	31	3	75 %
FPU_IP2 [15]	23	5	FPU	2889	187	50	231	2421	89	5	46	4	80 %
FPU_IP3 [16]	12	13	FPU	3720	226	63	277	2344	43	4	40	5	41 %
AMBER_IP [17]	48	8	ARM2	10342	695	280	742	6501	779	15	86	7	87.5 %
TIMER [18]	25	6	Peripheral	2919	19	163	292	1750	9	2	11	2	33 %

TABLE V
RTL ANALYSIS RESULTS

RTL Design Information			RTL Rule-base Analysis								Back Annotation (Spec & RTL mismatches)		
IP Name	entities	processes	clips facts for rtl	rules fired	functions inferred	fsm inferred	memory ram/rom	bus interface	Id coverage	average CF	IO based	register based	function inference mismatch
FPU_IP1	11	102	3335	92	4	0	0	0	57 %	0.48	0	NA	2
FPU_IP2	13	48	3965	162	5	1	0	0	71 %	0.93	0	NA	0
FPU_IP3	7	22	3919	73	4	0	0	0	53 %	0.7	0	NA	0
AMBER_IP	14	30	15195	269	15	2	2	1	85 %	0.9	2	0	0
TIMER	5	11	1232	24	2	0	0	1	90 %	0.72	1	9	0

Table IV provides spec analysis results. The table lists (a) spec extractor output as number of words extracted from XML version of each spec, and (b) their classification as spec ontology, english dictionary word etc. Then spec rule-base performs location and function analysis. The table also lists the number of functions and features inferred for each spec, and the spec coverage metrics. Spec coverage provides feedback to the user about spec sections of the untrusted IP that have no correspondences and need to be carefully reviewed. A low spec coverage also acts as feedback to KB developer to add more rules and to improve the spec ontologies.

Table V provides RTL analysis and back annotation results. The RTL to CLIPS translation was automatically performed for each IP. The table lists number of RTL clips facts, rules fired and inferences done for each IP. The function inferences refer to the number of behavioral functions being successfully inferred, e.g. FPADD in FPU domain. The bus-interface lists the number of protocol buses inferred in the IP. The IP level CF is the weighted average CFs of its lower level modules. The Id coverage corresponds to behavioral functions inferred for each IP in comparison to their number in PBM. A high CF means that functions along with low level properties get identified in RTL. The CF of the IP drops if there is a function inference mismatch between spec and RTL. The FPU_IP1 has two functions (Float to integer and Integer to Float) listed in its spec but not inferred from RTL. On manual review this discrepancy was confirmed. The register discrepancy for timer is RTL coding quality issue where register defines for address and name used in Spec are not found in RTL. The timer has a port naming discrepancy between specs and RTL. The ARM2 design has two extra ports in RTL as compared to Specs.

VI. CONCLUSION

We have presented a knowledge-guided methodology to perform analysis of Soft-IPs. Our approach can assist engineers in discovering discrepancies between IP spec and RTL. The coverage results can be used as metrics for IP analysis.

ACKNOWLEDGEMENT

This work has been supported by the DARPA IRIS Program under contract: HR001-11-C-0091. Rockwell Automation collaborated as a team effort in this project.

REFERENCES

[1] M. Keating and P. Bricaud, *Reuse Methodology Manual: For System-on-a-chip Designs*, 3rd ed. Boston, MA:Kluwer, 2002.
[2] J. Bhadra, M. S. Abadir, L. C. Wang, S. Ray. *A Survey of Functional Verification through Hybrid Techniques*, IEEE Design & Test of Computers, March-April 2007.
[3] A. Holt, E. Klein, *A semantically-derived subset of English for hardware verification*, Proc. 37th Annual meeting of Assoc. for Computational Linguistics, 1999.
[4] W. Cyre, *Capture, Integration, and Analysis of Digital System Requirements with Conceptual Graphs*, IEEE Trans. on Knowl. and Data Eng., 1997.
[5] W. Li, Z. Wasson, S. A. Seshia, *Reverse Engineering Circuits Using Behavioral Pattern Mining*, HOST 2012.
[6] P. Subramanyan, N. Tsiskaridze, *Reverse Engineering Digital Circuits Using Functional Analysis*, DATE 2013.
[7] K. Werner, *Can IP quality be objectively measured?*, DATE, 2004.
[8] L. Wang, H. Luo, *Automated IP Quality qualification for efficient SoC design*, Electronic Packaging Tech. and High Density Packaging (ICEPT-HDP), 2012.
[9] F. Wickberg, *HDL code analysis for ASIC in Mobile Systems*, *liu.diva-portal.org/smash/get/diva2:24142/FULLTEXT01*
[10] J. C. Giarratano, G. D. Riley, *Expert Systems: Principles and Programming*, 4th ed.: Thomson Course Technology, 2005.
[11] IEEE, *IEEE 754-2008 Standard for Floating point arithmetic*.
[12] S. Nirenburg, V. Raskin *Ontological Semantics* Cambridge, MA: MIT Press 2004.
[13] C. D. Manning et.al, *Introduction to Information Retrieval*, Cambridge University Press. 2008
[14] R. Usselmann, *http://opencores.org/project,fpu*.
[15] A. E. Jidan, *http://opencores.org/project,fpu100*.
[16] D. Lundgren, *http://opencores.org/project,fpu_double*.
[17] S. Cornor, *http://opencores.org/project,amber*.
[18] R. Hayes, *http://opencores.org/project,pit*.