Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 7 (2013) 391 – 397

Procedia Technology

The 2013 Iberoamerican Conference on Electronics Engineering and Computer Science

Automatic Verification of Assembling Digital Circuits by means of Semantic Web Techniques

Francisco E. Castillo-Barrera[a,*], Reyna C. Medina-Ramírez[b], Carlos Soubervielle-Montalvo[a], Marcela Ortíz-Hernández[a]

[a]*Engineering Faculty, Universidad Autónoma de San Luis Potosí, Dr. Manuel Nava 8, Zona Universitaria Poniente, C.P. 78290, San Luis Potosí, S.L.P., México*

[b]*Department of Electrical Engineering, Universidad Autónoma Metropolitana-Iztapalapa, Av. San Rafael Atlixco 186, Col. Vicentina, Iztapalapa, C.P. 09340, Distrito Federal, México*

Abstract

According to the last years the use of domain ontologies has increased considerably. In Logic Circuits, we can used the ontologies not only for educational purposes but also for interrogating the domain knowledge represented in an ontology. As well as means to verify the design of a circuit considering the manufacturer specification (offering) and the client view point (requiring). This approach allows the reuse of previously constructed circuits in different contexts.

Keywords: OWL-DL, Description Logics, One-bit full adder, Ontology, Semantic Technique.

1. Introduction

In the process of assembling a circuit using other existing circuits is necessary to verify that the circuits candidates meet the requirements requested. This is possible to express it as a contract between a supplier and a customer. Despite the existence of formal methods to verify assemblying, the reuses of circuits are still scarce. Formal methods have not been successful in the industry, even though they are the most reliable. This is due to the fact that high levels of training are required for its application, and its use often consumes more days of those assigned to this phase of the project.

The paper is structured as follows: Section 2 shows the state of the art. Section 3 introduces the concepts of techniques used in the Semantic Web. Section 4 shows an overview of descriptive and logical semantic approach to the verification of logic circuits. Section 5 describes the case of one-bit full adder. Section 6 describes how ontologies are used for the verification of a logic circuit. Finally, in Section 7 are the conclusions of the work.

[*]Corresponding author. Tel: +52 (444) 8262330 ext. 2339
Email address: `edgarcastillobarrera@gmail.com` (Francisco E. Castillo-Barrera)

doi:10.1016/j.protcy.2013.04.049

2. Related work

In the search for methods to verify contracts among logic circuits, we found a doctoral thesis related with formal methods by [1], in which the author makes an explanation of existing formal methods to verify proper assembly logic circuits. In his description indicates the use of mathematical proofs in its application, it should be noted that this requires a learning experience making them and mathematical proofs, our proposal does not require any mathematical learning or some sort in order to be adopted by the industry. Other related work was made by Burch et al [2], where the authors propose a symbolic version of the verification model known as "Model Checking" for verification of sequential circuits. But again, this proposal requires mathematical background which can be an impediment to their use in industry. Among the most related in the literature on ontologies and logic gates was made by [3] of which is the proposed use for the creation of intelligent learning objects aimed at being used for teaching basic logic gates[1].

3. Semantic Web Techniques

The *Semantic Web* [4][5][6] is an extension of the World Wide Web created by the british scientist Tim Berners-Lee who defines it as *"a web of data that can be processed directly and indirectly by machines"* [7]. This is a collection of standards, a set of tools [8], and a community that shares data. *Semantic Technology* is a concept in computer science which goal is to give semantics to data[9]. Supported by semantic tools [10] that provides semantic information about the meaning of words (RDF, SPARQL, OWL, and SKOS).

3.1. Description Logics (DL) and OWL

The description logics (DLs) are a family of logic-based formalisms for representing knowledge describing the domain in terms of concepts (classes), roles (relationships) and individuals. They are distinguished by:

- Formal semantics (model theoretic)
- Decidable fragments of First-order logic
- Provide services inference
- Decision procedures for robust and complete key problems (satisfiability, subsumption)
- Systems implemented (optimized)

3.2. Ontology consistency using Pellet an OWL-DL reasoner

Pellet [11] is an open-source Java based OWL-DL reasoner. In our verification process we use Pellet for checking the consistency of the ontology and classify the taxonomy. Pellet gives explanation when an inconsistency was detected. Restrictions can be expressed into an ontology. For instance, the following code verify that the logic circuit has at least 2 *xor* gates.

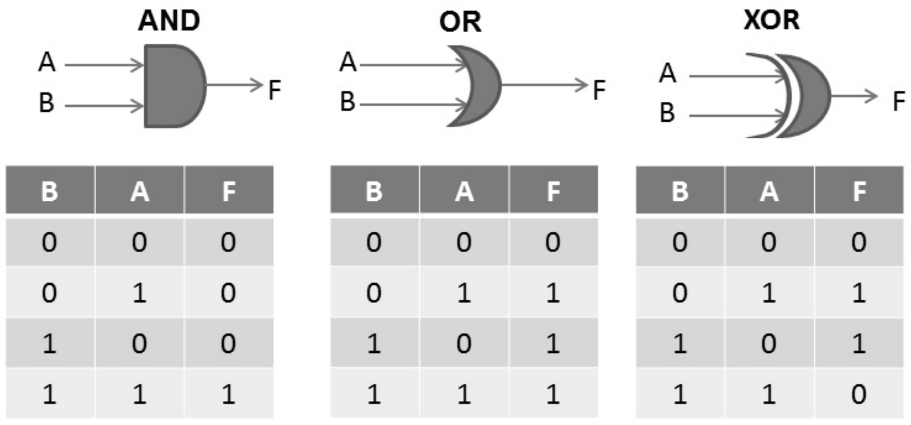
```
:Circuit rdfs:subClassOf
    [ a owl:Restriction ;
      owl:onProperty :hasXORgate ;
      owl:cardinality 2 ].
```

All properties defined in the Ontology are checked by the reasoner (Pellet) during the consistency verification process.

[1]The AND, OR and NOT gates are considered as basic gates.

4. Logic Circuits Verification: a Semantic Approach

Semantic verification is the process which uses an *Ontology*, *Reasoners* and *SPARQL* queries to guarantee the correct construction of logic circuits with specific connections and outputs. The semantics of assembling the logic gates are described with object properties and methods. An important aspect of the logic gates to consider during the assembling is the number of inputs and Outputs connections. A logic gate has one output, but different number of input connections. The logic gate connections are based on the output of one of them using as input in the others. The graphic representation and truth table of the basic logic gates is showed in figure 1.



B	A	F
0	0	0
0	1	0
1	0	0
1	1	1

B	A	F
0	0	0
0	1	1
1	0	1
1	1	1

B	A	F
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 1. Logic gates required by the FullAdder

5. 1-Bit Full Adder

A full adder is digital cirtcuit which adds binary numbers and accounts for values carried in as well as out, as shown in the figure 2. A one-bit full adder adds three one-bit numbers, with three inputs A, B, and Cin; A and B are the operands, and Cin is a bit carried in from the next less significant stage. The circuit produces a two-bit output sum typically represented by the signals CARRY and SUM. The one-bit full adder's truth table based on instances and using *n3* notation [12] is showed in table 1 :

Table 1. 1-Bit Full Adder Truth Table based on Instances and n3 notation

Input A Input B	CIN	CARRY	SUM
:0_0	:0	:0	:0
:0_0	:1	:0	:1
:0_1	:0	:0	:1
:0_1	:1	:1	:0
:1_0	:0	:0	:1
:1_0	:1	:1	:0
:1_1	:0	:1	:0
:1_1	:1	:1	:1

The one-bit full adder was chosen as a case of study due to its high modularity, it means that these circuits could be connected between them in several ways allowing the assembling of more complex arithmetic circuits. For example connecting **n one-bit** full adders in cascade it is possible to assemble a **n-bits adder**, and even more reusing this **n-bits adder** and connecting in the correct way it is possible to assemble a **n-bits subtractor**.

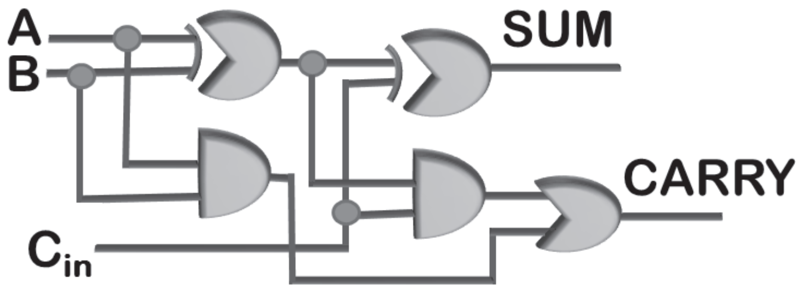


Fig. 2. 1-Bit Full Adder Circuit

6. A Core Circuit Ontology in OWL-DL

We proposed a core ontology which has the minimum concepts contained in the digital circuits ontology analyzed in this work. A core Ontology is built by means of classes and relations among concepts. These concepts and classes correspond to the specification of an abstract data type and a set of methods that operate on that abstract data type. The typing information describes the types of input and output or both bit variables. All of the above is represented in our ontology (class LogicGate, Circuit and Bits). The Ontology classes are showed below:

```
(1) :Circuit   a owl:Class .
(2) :LogicGate    rdfs:subClassOf :Circuit ;
(3)              rdfs:label "A logic gate is an
(4)                    device implementig a Boolean function" ;
(5)              rdfs:comment "It performs a logical operation".
(6) :Bits      a owl:Class ;
(7)             rdfs:label "Bit means Binary Digit" ;
(8)             rdfs:comment "1 bit is equal to 0 or 1".
```

The ontology is populated with basic logic gates. An example of "and" logic gate definition using the ontology language is showed below:

```
(1) :and   a :LogicGate ;
(2)      rdfs:label "AND gate" ;
(3)      rdfs:comment "0 0 =  0";
(4)      rdfs:comment "0 1 =  0";
(5)      rdfs:comment "1 0 =  0";
(6)      rdfs:comment "1 1 =  1".
(7) :and1 a :Gate .
(8) :and1  owl:sameAs :and .
```

In line 8 a new instances *and1* is created with the same characteristics as *and* only using the *Owl* keyword *sameAs*.

6.1. Verification using Semantic Queries in SPARQL

We decided to explore semantic queries in *SPARQL* instead of production rules [13]. The second step after the reasoner have checked the ontology consistency is to apply a SPARQL query. We decide to define a specific query which evaluates and verifies certain information on the input model. Of course, all this process is transparent, for the user. We have used Jena API [14] and Java language [15] for programming that and NetBeans IDE 7.0 [16]. *SPARQL* is the version of *SQL* for ontologies. But, we can use variables in the queries, constraints, filtering information, logic operators, if statements and more. Lines are linking by variables which begin with a question mark. The same name of variable imply the same value to look for in the query. The *Jena* API allowed us to use *SPARQL* queries in our framework programmed in *Java* language. Part of the source code from the query is showed below:

```
SELECT DISTINCT ?Interface1 ?Interface2 ?Match_Method
               ?Match_Precond
{
  ?Interface1  :typeInterface        :required ;
               :hasMethod            ?Method1 .
  ?Method1     :hasParameter         ?par1 ;
               :hasMethodName        ?name1 ;
               :hasNumParameters     ?numpar1 .
  ?par1        :hasIndexOrder        ?pos1 ;
               :hasDataTypeParameter ?partype1 .

 ?Interface2  :typeInterface        :provider .
 ?Interface2  :hasMethod            ?Method2 .
 ?Method2     :hasParameter         ?par2 ;
              :hasMethodName        ?name2 ;
              :hasNumParameters     ?numpar2 .
 ?par2        :hasIndexOrder        ?pos2 ;
              :hasDataTypeParameter ?partype2 .

  BIND(if( ?name2 = ?name1, fn:substring(?name1,1,
                       fn:string-length(?name1))
          , "No match name") AS ?Match_Method )
                       :
} order by ?Match_Method
```

6.2. *Ontology-Driven Translator*

The process before to apply *SPARQL* queries require that contracts are written using *OWDL-DL* language, for do that we have programmed an Ontology-Driven Translator which receives a specification in IDL files (Interface Definition Language) and translate in *OWL-DL* code. The contracts are showed in the source code below:

```
module BasicLogicGates
{     domain          Circuits;
      subdomain     LogicGates;
      post: and == 1.
      post:  or == 1.
      post: xor == 1.

      provided interface ILogicGates{
            void and(in short A, in short B, out short F);
            void or(in short A, in shor B, out short F);
            void xor(in short A, in short B, out short F);
      };
};
```

```
module HalfAdder
{   domain     Circuits;
    subdomain  Adders;
    pre: xor == 1.
    pre: and == 1.
    post: halfAdder = 1.
```

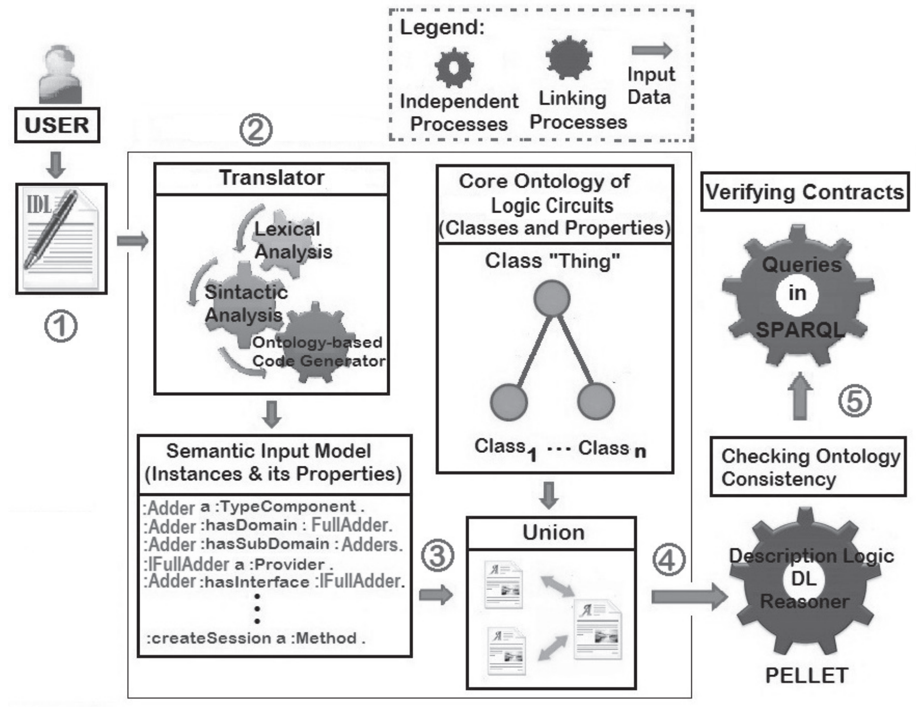

Legend:
Independent Processes
Linking Processes
Input Data
USER
IDL
①
②
Translator
Lexical Analysis
Sintactic Analysis
Ontology-based Code Generator
Semantic Input Model (Instances & its Properties)
:Adder a :TypeComponent .
:Adder :hasDomain : FullAdder.
:Adder :hasSubDomain : Adders.
:IFullAdder a :Provider .
:Adder :hasInterface :IFullAdder.
:createSession a :Method .
Core Ontology of Logic Circuits (Classes and Properties)
Class "Thing"
Class1 ... Class n
③
Union
④
Verifying Contracts
Queries in SPARQL
⑤
Checking Ontology Consistency
Description Logic DL Reasoner
PELLET

Fig. 3. Semantic process using an Ontology-Driven Translator

```

	provided interface IHalfAdder{
	  void halfAdder(in short A, in short B, out short C, out short S);
	};
	required interface IFullAdder{
		void xor(in short A, in short B, out short AxorB );
		void and(in short A, in short B, out short AandB );
	};
};
```

The whole process is showed in figure 3.

7. Conclusions

Automatic Verification of Digital Circuits in *OWL-DL* is possible using *domain ontologies*, *reasoners*, *ontology-driven translators* and *SPARQL* queries in every part where logic gates are used. Ontologies are usually expressed in a formal logic-based language (Description-Logic) and are composed by classes, properties and relations. Ontologies give more expressive meaning, but maintains computability. For an Automatic Verifcation of Digital Circuits based on Semantic Web Techniques, it is neccesary to write IDL files with specification of contracts (logic gates required for building the circuit). In this paper we have presented this approach and show some benefits such as: reuse of previously constructed circuits in different contexts reducing assembling time and cost. It is important to mention that this work is being taken as a case study based on a combinational circuit such as the basic full adder, because it will allow us in the future to design more complex circuits with different functionalities. And thus testing the proposed verification process in more complex combinational circuits.

References

[1] C. van Eijk, Formal methods for the verification of digital circuits, Technische Universiteit Eindhoven, 1997.
[2] J. Burch, E. Clarke, K. McMillan, D. Dill, Sequential circuit verification using symbolic model checking, in: Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE, IEEE, 1990, pp. 46–51.
[3] T. Robal, T. Kann, A. Kalja, An ontology-based intelligent learning object for teaching the basics of digital logic, in: Microelectronic Systems Education (MSE), 2011 IEEE International Conference on, IEEE, 2011, pp. 106–107.
[4] W. P. Davies John, Stunder Rudi, Semantic web technologies trens and research in ontology-based systems (2006).
[5] K. Breitman, M. A. Casanova, W. Truszkowski, Semantic Web: Concepts, Technologies and Applications (NASA Monographs in Systems and Software Engineering), Springer-Verlag London, 2006.
[6] K. T. S. Michael C. Daconta, Leo J. Obrst, The Semantic Web: A guide to the future of XML, Web Services and Knowledge Management, Wiley Computer Publishing, Inc., 111 River Street Hoboken, NJ, 2003.
[7] T. Berners-Lee, J. Hendler, O. Lassila, Others, The semantic web, Scientific American 284 (5) (2001) 34–43.
[8] W. M. K. B. Duineveld A.J., Stoter R., B. V.R., Wondertools? a comparative study of ontological engineering tools (2000).
[9] T. Gruber, Ontolingua: A mechanism to support portable ontologies, `http://ontolingua.stanford.edu` (1992).
[10] S. Staab, A. Gómez-Pérez, W. Daelemana, M.-L. Reinberger, N. Noy, Why evaluate ontology technologies? because it works!, Vol. 19, 2004, pp. 74–81.
[11] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical owl-dl reasoner, Web Semantics: science, services and agents on the World Wide Web 5 (2) (2007) 51–53.
[12] T. Berners-Lee, N3 notation: http://www.w3.org/designissues/notation3.html.
[13] A. C. del Río, J. E. L. Gayo, J. M. C. Lovelle, A model for integrating knowledge into component-based software development, KM - SOCO (2001) 26–29.
[14] Jena, Jena a semantic web framework for java, `http://jena.sourceforge.net/` (2000).
[15] P. J. Clarke, D. Babich, T. M. King, B. M. G. Kibria, Model checking and abstraction, ACM Transactions on Programming Languages and Systems 16 (1994) 1512–1542.
[16] T. Boudreau, NetBeans: the definitive guide, O'Reilly Media, 2002.