Towards a method for VLSI circuit reverse engineering

Hamza Bouchaour, Mohammed Ouali, Yahia Lebbah
LITIO Lab
B.P. 1524 EL-M'Naouar
Oran, 31000, Algeria
bouchaour@hotmail.fr, mohammed.ouali@univ-oran.dz, ylebbah@gmail.com

***Abstract*—This paper tackles the VLSI circuit reverse engineering problem. Actual VLSI circuits are made of several millions of transistors or hundreds of thousands of logical gates. Whether it be for circuit verification, functional abstraction, or simply circuit understanding, reverse engineering aims at building a hierarchy from the transistor level, to gate level, to register level, up to more complex components. The idea is to transform the circuit into a target graph and to look for instances of sub-circuits as subgraph patterns in the original graph. In this work, we propose to cast the subgraph isomorphism as a constraint satisfaction problem, where an isomorphism is expressed by a set of constraints and filters.**

I. Introduction

In this paper, we are interested in reverse engineering of digital VLSI circuits. In the standard cell-based digital circuit design [1], the term *netlist* refers to the graph that describes the connectivity of VLSI design. A digital circuit is generally designed using automated design tools, combined with hardware description languages to generate a transistor *netlist*. In many situations, one might need to guess and find the functionality of a circuit from its transistor *netlist*. To recover the original design through the reverse engineering of circuit design, it is necessary to process the transistor's *netlists* in order to reach some more meaningful levels of abstraction (transistors, gates, Inputs). Since the electronic circuits are always represented as graph structures, therefor, there is a need for graph pattern matching techniques that can identify groups of transistors or gates based on their electrical connections, rather than their physical layouts. Our main contribution in this paper is to treat the sub-circuits extraction as a subgraph isomorphism problem (SIP) [2]. SIP is a well known NP-complete problem in general, and most studied in graph theory research [3]. In other words, the run time to detect a subgraph isomorphism between two graphs is, in the worst case, exponential to the number of nodes of these graphs. Subgraph isomorphism has already been used in circuit design. In VLSI computer-aided design (CAD) [4], one of the critical problems is that of determining whether the layout of the circuit geometry corresponds to the requirements of the circuit [5], especially to find the isomorphic sub-circuits from a lager circuit using the process of reverse engineering. In this paper, we mainly focus on this issue. We consider a circuit for which we ignore the functionality and hence the components that it is made of. This circuit is represented by a target graph. We build a pattern graph incrementally from elementary components (transistors than gates, to registers). Then, we look for instances of a pattern graph in the target graph [6]. In general, according to [7], algorithms for subgraph isomorphism problems fall into two major categories:

- **Dedicated algorithms**: based on tree search approach as *Ullmann* [8] and *Vf lib* presented in [9], these algorithms quickly become ineffective when graph size becomes larger, as predicted by the NP-completeness of SIP. Thus, for some problems as sub-circuit verification, the dedicated algorithms became obsolete.
- **Constraint programming (CP)**: is the best way to process this problem, since SIP is a classical constraint satisfaction problem. This method provides a declarative setting for the resolution. It induces the use of a generic constraint solver. Among the most recent works in this field, we cite *Zampelli* and *Deville* [10] who developed an iterative filtering algorithm, based on the notion of extended neighborhood. Recently, *Solnon* [11] has improved this algorithm with a much stronger filtering procedure.

Here, we opted for the constraint programming (CP) strategy to solve the subgraph isomorphism problem. To do this, we took back the main idea that was proposed in [10], which we tried to adapt to our problem. We elaborate a resolution model for the subgraph isomorphism problem formulated as a constraint satisfaction problem, where the structure of the pattern sub-circuit is translated into structural constraints that are resolved by a constraint solver. We propose to this model two configurations that use the edges-list for graph representation. The remainder of this paper is organized as follows. Section 2 provides relevant background and discusses related work on reverse engineering of sub-circuits. Section 3 describes the CSP model that we propose for the resolution of the subgraph isomorphism problem. It also describes the two configurations of this model. The experimental results derived from the implementation of the CSP model are reported and discussed in section 4. Section 5 concludes the paper and outlines possible future work.

II. Background and related work

In this section, we review necessary background and discuss relevant work.

A. Netlist

A netlist is a list of electrical components of a circuit and their interconnections, generally represented as a *hyper graph*. Each interconnection or "net" is assigned a unique label. The netlist will list each component as well as every net to which each of the terminals of each component is connected.

B. Isomorphism

Isomorphism is defined as having the "same form" or the "same shape". If two groups of elements are isomorphic, there is a one-to-one relationship between the elements of one group and the elements of another. Graph isomorphism signifies that the two entire graphs are identical. Sub-graph isomorphism implies that there is a one-to-one relationship between each element of a small graph and the corresponding sub-graph of another larger graph. Here, a small graph means a graph with few nodes.

C. Sub-circuit extraction problem

An example of the sub-circuit extraction problem is shown in Figures 1 and 2, where one tries to pull out a 1bit full adder from a much bigger circuit. The sub-circuit extraction problem is whether or not there is any instance of one-bit full adder in the main circuit. Furthermore, if there are some, how many? The problem of sub-circuit extraction can be transformed to a subgraph isomorphism problem. Given a graph G, a subgraph S, has all its nodes and its edges in G. The subgraph isomorphism detection can be defined as: Given a graph S and a larger graph G, find all the subgraphs of G that are equivalent to S. Similarly, the sub-circuit extraction problem is to identify all the sub-circuits in the main circuit that are equivalent to the pattern circuit. The subgraph isomorphism is a reliable technique of pattern matching that may be used advantageously in this paper.

D. An illustrative Example

We present in this section an applicative example of the sub-circuit extraction in the VLSI context, in order to spot the functionality of the main circuit. By convention, the circuits used in our implementation are from the same type, *i.e.* they achieve the same elementary function. The sub-circuit that we chose to extract, called the pattern graph, is the one-bit full adder with carried propagation. This circuit is made of 18 transistors and 21 nets. The goal is to find the number of instances of the pattern graph inside the target graph using the principle of subgraph isomorphism, and identify each instance in the target graph.

1) First example: Figure 1 describes the search of a one-bit full adder inside five one-bit full adders. The correct outcome is to find five instances of graphs that fulfill the isomorphism condition.
Our tool manages to solve this instance by identifying the five one-bit full adders, described by their main inputs and outputs, that allow to recognize the functionality of the circuit that is the binary addition of five bits.

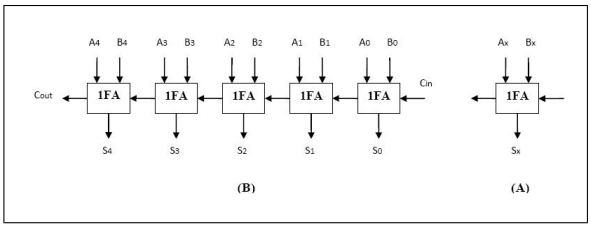

Fig. 1. (A):1bit full adder, (B):Five 1bit full adders.

2) Second example: Figure 2 describes the search of a one-bit full adder inside three two-bits full adders. The goal is to find six instances of graphs that fulfill the isomorphism condition.

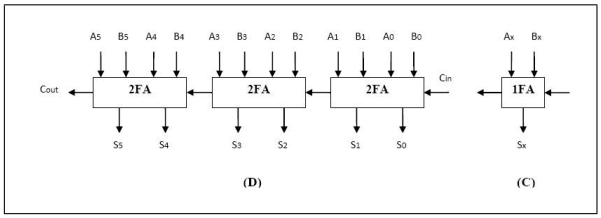

Fig. 2. (C):1bit full adder, (D):Three 2bits full adders.

Our tool manages to solve this instance by identifying the six one-bit full adders, described by their main inputs and outputs, that allow to recognize the functionality of the circuit that is the binary addition of six bits.

E. Related Work

Research in reverse engineering of digital VLSI circuits started in the early 1990s with a main focus on formal methods. Traditional simulation verification methods became no longer adequate to circuit proprieties as complexity and size, *i.e.* a 8-bit counter takes just 256 vectors to fully test the state space, but a 32-bit counter takes over 4 billion vectors to test the state space. So, the number of vectors required to fully simulate a design made it impractical to use simulation to verify synthesized logic designs, besides the fact that these techniques relied on the specific characteristics of the technology or circuits being extracted, and did not generalize to allow arbitrary sub-circuits to be found, such as analog circuits. Treating the sub-circuit extraction as a subgraph isomorphism problem was the best alternative to achieve a true technology-independent solution. Algorithms of subgraph isomorphism can be used in many different contexts. They include digital and analog circuits, using varying levels of abstraction. Authors of [12] were the pioneers to solve the sub-circuit extraction problem based on a solution to the subgraph isomorphism. They proposed an algorithm which has been implemented in a commercial software entitled *SubGemini*. In this method, the pattern and the main circuit are labeled alternately. Labels are based on matched neighbors. Thus, if two nets have the same label, a match is possible. We opted in the present work to the *Zampelli* [10] technique to resolve the subgraph isomorphism problem while matching not nets

but edges between two adjacent devices, a technique which we believe is the more efficient approach. Our method is dedicated to recover the functionality of circuits and will be amply presented in the following section.

III. Solving approaches

To achieve the reverse engineering of digital VLSI circuits, in the context of the subgraph isomorphism problem, we first introduce the specific data structure we use to represent the graphs. Then, we express the wording of the subgraph isomorphism problem as *CSP* model. Note that here we pick up the outlines of the model that has already been developed by *Zampelli and al* in [10], since our goal in this paper was not to resolve the subgraph isomorphism problem but rather to adapt the best solution to this problem for the sub-circuit extraction process. At last, we describe minutely the implementation that we propose to the *CSP* model which differs slightly from the *Zampelli and co* algorithm.

A. Modeling the input circuit

In this paper, we use the structure of graphs to represent the internal architecture of the circuits. Among several possibilities that exist in graph theory to represent a simple graph, we chose to use the representation as *"edges list"* in order to minimize the amount of data to be stored in memory. This structure permits to characterize a graph by only storing its respective edges. This results in dramatically reducing the amount of data to store in memory. To construct these edges lists, we use an intrinsic property of the constraint solver called *Gecode* [13], that supports this type of structure. Each edge of the target graph (which represents the set of candidates edges) is stored in a tuple that contains the identifiers of the two ending vertices. In our *Gecode* implementation, the edges are stored in a special structure called *TupleSet*.

B. Formulation as a constraints satisfaction problem (CSP)

The subgraph isomorphism problem between a pattern graph $G_p(N_p, E_p)$ and a target graph $G_t(N_t, E_t)$ (where N_p and N_t refer to the set of respective pattern and target nodes, E_p and E_t refer to the sets of respective pattern and target edges) can be formulated as a CSP model in a very simple way. A variable x_u is associated to every node u of the pattern graph. Thus, we'll get as many variables as nodes in the target graph. The domain of every variable is the set of nodes from the target graph. Then, we lay two constraints on these variables that ensure the matching of edges of the two graphs. We describe in the following the CSP model.

The CSP model: Let the *CSP* model of the subgraph isomorphism problem defined by the triplet (X, D, C). This triplet is defined as follows:

- $X = \{x_1, \ldots\ldots, x_{np}\}$: is the set of the variables that are assigned to the pattern graph nodes.
- $D = D_1 = D_2 = \ldots = D_{np} = \{1, \ldots, n_t\}$: it represents the variable's domain, the domain of every variable is the set of the target graph nodes.
- $C = \{C_1(x_1, \ldots, x_{np}), C_2(x_1, \ldots, x_{np})\}$: it represents the two constraints submitted to the model which are:

$C_1(x_1, \ldots, x_{np})$ = *AllDiff* $(x_1, \ldots, x_{np})$. C_1 imposes that every node from the pattern graph is only matched to one node of the target graph. It assures that the matching function is bijective.
$C_2(x_1, \ldots, x_{np})$: $\forall$ (u, v) $\in N_p \times N_p$, c2(x_u,x_v) $\equiv$ ((u,v) $\in E_p \Rightarrow (x_u,x_v) \in E_t$). C_2 imposes that every edge from the pattern graph has a support in the target graph. We note that n_p refers to the number of pattern graph nodes, and n_t refers to the number of target graph nodes. This concerns the edges of the pattern and the target graphs. We present in the following the implementation that we elaborated to the *CSP model*.

C. Implementation of the CSP model

We propose to the *CSP model* a robust implementation. It includes some *integer variables* in order to represent the data, and two constraints: the first one is a *difference* constraint. The second one is an *isomorphism* constraint that checks the edge matching between the two graphs. The latter is set by the well known CP constraint called *extensional constraint*. The implementation also includes a *"pre-processing procedure"* with respect to the variable's domains. This pre-processing step corresponds to the basic level of the *ILF* algorithm [10]. In this algorithm, the *pre-processing procedure* is run during the filtering step. In our implementation, we choose to run this procedure upstream of the filtering operation which could affect the overall performances of the implementation. We supply to this implementation two different configurations that also may affect the performances of each one of them. In the following, we describe the highlights of the *pre-processing procedure* and its both configurations.

1) The pre-processing procedure: The role of this procedure is to minimize the domains of variables before they're considered in the resolution process. This pre-processing step is widely discussed in [10]. This procedure greatly reduces the number of allowed values for each variable. Consequently, the CSP model performance may get improved especially in the running time. The pre-processing procedure uses a widely-known property in graphs theory that is the notion of labeling nodes. The idea is to associate to each node of the graph an invariant property as the node degree, then to define a partial order between these labels. Thus, a node v of the target graph can be matched to a node u of the pattern graph only if the pair (u, v) satisfies the partial order. In this case, the nodes *u* and *v* are called compatible. In this paper, we just reproduce the basic level of the *ILF* algorithm. After this step, the search of the subgraph isomorphism continues with variables whose domains have been reduced.

2) The pre-processing CSP model: The enforcement of algorithm1 that implements the *pre-processing CSP model* is as follows: For each edge (u, v) of the pattern graph, *TupleSet*

provides the set of edges that can be matched to this edge. Formally, the constraint "*Extensional* ((u, v), TupleSet)" is going to put the fact that the target edge (x_u, x_v) matches the pattern edge (u, v) (it exists in the target graph whose edges are stored in TupleSet). The *pre-processing CSP model* is shown in algorithm 1. Note that `(u, v)` refer to the edge belonging to the pattern graph, and `TupleSet` provides the set of the candidate edges of the target graph.

Algorithm 1 pre-processing CSP model

C_1:alldiff $(x_1, \ldots, x_{np})$.

for each node u $\in N_p$ **do**
  D(u) $\Leftarrow$ D(u) $\cap$ {v $\in N_t$ — deg(u) $\leq$ deg(v)}
end for

for $u = 0$ to $u = n_p$ **do**
  for $v = 0$ to $v = n_p$ **do**
    if (u,v) $\in E_p$ **then**
      C_2: **Extensional**((u,v), TupleSet).
    end if
  end for
end for

3) Optimized pre-processing CSP model: In this implementation, we use a different configuration of the *pre-processing CSP model* that may specially improve the running time, but at the cost of an enough excessive use of the memory. In the first model shown in III-C1, we use for each node of the pattern graph a temporary array that contains the whole of its compatibles nodes. Then, this array is assigned to the definition domain of this pattern node. For the following pattern node, this array will not be drained. It'll contain the whole of compatibles nodes of the second pattern node, plus the set of compatibles nodes of the previous pattern node. Then, as previously, this updated array will be assigned to the definition domain of the second pattern node. Arriving at the last pattern node, the array will contain all nodes from the target graph that are compatibles with those of the pattern graph. In the *optimized pre-processing CSP model*, the temporary array is drained after each assignation of the definition domain to the corresponding pattern node. By this way, the definition domain of each node will only contain the set of nodes that are compatible with this one, which will dramatically reduce the number of candidate nodes in the matching process of each pattern node. Consequently, there will be fewer associations and therefore a better running time of the constraint's solver. In return, the memory usage will be very excessive due to the draining operations performed after each new assignment of the definition domain for each pattern node, which requires many treatments and therefore a large memory allocation. We'll see in the next section a comparative survey between the performances of both configurations to the pre-processing models that have already been discussed.

IV. Experimentation

The pre-processing CSP model was implemented on the *Gecode*[13] C++solver. The experiments were run on a Windows laptop with Intel Core2Duo CPU and 2GB RAM. The experiments have the goal of finding structural isomorphism in large designs. To generate interesting examples, we generated synthetically different types of only full adders ranging from 1bit to 32bits giving us a large panel of graphs. We considered in the experimentations two different instances of graphs: the first instance included a one-bit full adder regarded as the pattern graph, and several one-bit full adders regarded as the targets graphs. The second instance also included a one-bit full adder regarded as the pattern graph, and this time several two-bit full adders regarded as the targets graphs. We also fixed a maximum running time (**Max**) to 2 hours and 30 minutes. In the following, we first present the results obtained by the *pre-processing CSP model* called (CSP+P), followed by the results got by the *optimized pre-processing CSP model* called (CSP+O).

A. Pre-processing CSP model (CSP+P)

Table I describes the performances of the (CSP+P) model in the search for an isomorphism between a one-bit pattern graph and several one-bit target graphs. The pattern graph contain 18 nodes (transistors). The first line of table I shows the number of adders used in the experiment, and considered as target graphs. These graphs contain 180, 1800, 9000, 27000 nodes (or transistors) respectively, since one full adder is composed of 18 transistors. These last are illustrated in the second line. The third line shows the running time (Run) in seconds. The fourth line shows the number of failed nodes (Fail), *i.e.* nodes in the search tree that do not lead to any solution in a *Depth-first search (DFS)* (DFS is a search algorithms used for graphs and trees. One starts at the root and explores as far as possible along each branch before backtracking). Line five shows the memory usage (Mem) in megabyte (MB).

TABLE I
1BIT PATTERN GRAPH AND ONE-BIT TARGETS GRAPHS

CSP+P	10	100	500	1500
Nodes	180	1800	9000	27000
Run	0.094	3.719	92	901
Fail	77	797	3997	11997
Mem	0.1 MB	0.786 MB	5.48 MB	12.65 MB

We notice that the (CSP+P) model solves quickly instances that contain a few adders, but running time degrades as the number of adder increases over 1000. Table II describes the performances of the (CSP+P) model in the search of an isomorphism between a one-bit pattern graph and several two-bits target graphs. The pattern graph also contain 18 nodes (transistors). The first line of table II shows the number of adders used in the experiment, and considered as target graphs. These graphs contain 360, 3600, 18000, 54000 nodes (or transistors) respectively. These last are illustrated in the second line. We use in the other lines the same table structure.
We notice that the (CSP+P) model generates very few failure nodes, thanks to the pre-processing procedure that permitted to reduce upstream the number of the values that can be assigned to the variables. We'll see in the second configuration that this number will be even smaller. When running the (CSP+P)

TABLE II
1BIT PATTERN GRAPH AND 2BIT TARGETS GRAPHS

CSP+P	10	100	500	1500
Nodes	360	3600	18000	54000
Run	0.266	13.75	378	2622
Fail	157	1597	7997	23997
Mem	0.2 MB	1.68 MB	8.67 MB	22.06 MB

model, 9175 seconds of running time (**Max**) allow to process 100008 nodes (transistors).

B. Optimized pre-processing CSP model (CSP+O)

Table III describes the performances of the (CSP+O) model in the search for an isomorphism between a 1 bit pattern graph and several 1 bit target graphs.

TABLE III
1BIT PATTERN GRAPH AND ONE-BIT TARGETS GRAPHS

CSP+O	10	100	500	1500
Run	0.063	1.625	36.84	353
Fail	0	0	0	0
Mem	0.1 MB	2.11 MB	36.81 MB	296.27 MB

Table IV describes the performances of the (CSP+O) model in the search for an isomorphism between a 1 bit pattern graph and several 2 bits target graphs.

TABLE IV
1BIT PATTERN GRAPH AND 2BIT TARGETS GRAPHS

CSP+O	10	100	500	1500
Run	0.156	5.18	156	941
Fail	0	0	0	0
Mem	0.3 MB	6.78 MB	133 MB	1219 MB

We notice from tables III and IV that the (CSP+O) model is in average 2.6 times faster than the (CSP+P) model, but it also consumes 30 times more memory. The other difference is that it generates no failure nodes thanks to the draining operations of arrays that ensure for each pattern node to have only compatibles nodes with it. The advantage of the (CSP+O) model is that it permits to process 67500 nodes (transistors) in only 1418 seconds. Beyond, this configuration cannot follow because of the memory usage problem briefly described in III-C3. In summary, regarding both configurations that we proposed to the CSP model, our preliminary implementations works reasonably well but it has not been fine-tuned in terms of quality of results and runtime. However, the results are encouraging, demonstrating that a basic reverse engineering of large circuits can be performed as a subgraph isomorphism problem.

V. CONCLUSION AND FUTURE WORK

The objective of this paper was to study the reverse engineering problematic of digital VLSI circuits, in the context of a subgraph isomorphism problem by the constraints programming approach. The problem consisted in recovering instances of a pattern graph in a much larger target graph and containing several instances of the pattern graph. The recent works of [10] were at the origin of our choice to use the constraints programming approach to solve this problem, as this technique is up to now the *state-of-the-art* method according to [10], [11]. Our contribution was a CSP model permitting to express the sub-circuits extraction as a subgraph isomorphism problem. The implementation that we proposed to the CSP model allowed us to process graphs of large size that contain 5556 one-bit full adders such in 2 hours and 32 minutes. This last result is enough promising. We propose, as a continuation of this work, the adaptation of the implementations so that it can deal with huger graphs in less running time (or to improve its scalability). Also, it would be interesting to exploit other types of combinational circuits in the process of sub-circuits extraction.

REFERENCES

[1] A. Richard, "Contribution à l'implmentation d'un flot de conception de circuits intgrés basè sur une bibliothque virtuelle," PhD thesis, Montpellier II University, 2003.

[2] H. Motoda, "Pattern discovery from graph-structured data - a data mining perspective," *Lecture Notes in Computer Science*, vol. 4570/2007, pp. 12–22, 2007.

[3] Garey and Johnson., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co, Jan. 1979, vol. 15.

[4] Z. Ling and al, "An efficient subcircuit extraction algorithm by resource management approach," in *Second International Conference On ASIC, Shanghai, P.R. China*, 1996, pp. 21–24.

[5] C. Ebeling and al, "Validating vlsi graph layout by wirelist comparison," in *the Conference on Computer Aided Design (ICCAD)*, 1983, pp. 172–173.

[6] Conte and Foggia., "Thirty years of graph matching in pattern recognition," *IJPRAI*, vol. 18(3), pp. 265–298, 2004.

[7] S. Zampelli, "A constraint programming approach to subgraph isomorphism," PhD thesis, Catholic University of Louvain, 2008.

[8] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of Insect Behavior*, vol. 23(1), pp. 31–42, 1976.

[9] Foggia and al, "An improved algorithm for matching large graphs," in *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*, ser. IAPR-TC15, 2001, pp. 149–159.

[10] Zampelli and al, "Solving subgraph isomorphism problems with constraint programming," *Constraints*, vol. 15, no. 3, pp. 327–353, Jul. 2010. [Online]. Available: http://liris.cnrs.fr/publis/?id=4328

[11] C. Slonon, "Alldifferent-based filtering for subgraph isomorphism," *Artificial Intelligence*, vol. 174(12-13), pp. 850–864, 2010.

[12] M. Ohlrich and al, "Subgemini: Identifying subgraphs using a fast subgraph isomorphism algorithm," in *30th ACM/IEEE Design Automation Conference*, 1993, pp. 31–37.

[13] Schulte and al, "www.gecode.org, Tech. Rep.