МАТЕМАТИЧЕСКИЕ ОСНОВЫ НАДЁЖНОСТИ ВЫЧИСЛИТЕЛЬНЫХ И УПРАВЛЯЮЩИХ СИСТЕМ

УДК 681.32 DOI 10.17223/20710410/64/4

GRAPH METHODS FOR RECOGNITION OF CMOS GATES IN TRANSISTOR-LEVEL CIRCUITS

D. I. Cheremisinov, L. D. Cheremisinova

The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, Belarus

E-mail: {cher, cld}@newman.bas-net.by

The paper focuses on the decompilation of a flat transistor circuit in SPICE format into a hierarchical network of logic gates. The problem arises in VLSI layout verification as well as in reverse engineering transistor circuit to redesign integrated circuit and to detect untrusted attachments. The most general case is considered when the extraction of functional level structure from transistor-level circuit is performed without any predetermined cell library. Graph methods for solving some key tasks in this area are proposed. The presented graph methods have been implemented in C++ as a part of a decompilation program, which has been tested using practical transistor-level circuits.

Keywords: *CMOS transistor circuit, subcircuit extraction, logic gate recognition, graph isomorphism, SPICE format.*

ГРАФОВЫЕ МЕТОДЫ РАСПОЗНАВАНИЯ КМОП-ВЕНТИЛЕЙ В СХЕМАХ ТРАНЗИСТОРНОГО УРОВНЯ

Д. И. Черемисинов, Л. Д. Черемисинова

Объединенный институт проблем информатики НАН Беларуси, г. Минск, Беларусь

Рассматривается задача декомпиляции плоского описания транзисторной схемы в формате SPICE в иерархическое описание схемы на уровне логических элементов. Проблема декомпиляции возникает при верификации СБИС путём сравнения исходного описания для синтеза транзисторной схемы со схемой, восстановленной из топологии, а также при обратном инжиниринге для перепроектирования интегральных схем и обнаружения несанкционированных вложений. Рассматривается случай, когда при извлечении структуры функционального уровня из транзисторной схемы библиотека исходных логических элементов не известна. Предложены графовые методы для решения некоторых ключевых задач, возникающих при декомпиляции описания транзисторной схемы. Представленные методы реализованы на языке C++ как часть программы декомпиляции, которая протестирована на практических схемах транзисторного уровня.

Ключевые слова: *КМОП-схема из транзисторов, экстракция подсхем, распознавание логических вентилей, изоморфизм графов, формат SPICE.*

1. Introduction

Currently CMOS is the dominant technology used in most very large scale integrated (VLSI) circuit chips: more than 95 % of integrated circuits are fabricated in CMOS. Modern digital CMOS circuits contain up to a billion primitive elements at the transistor level, and the complexity of systems rapidly increases while time-to-market is imposed to decrease. Rapid verification of the software implementations and the detection of logical errors are becoming a major bottleneck in VLSI Computer-Aided Design (CAD). One of the most important steps in CAD VLSI circuit is to ensure that the final layout of the circuit geometry correctly represents the intended logic of the previous specification. Traditional test method, such as switch-level simulation, is an effective means for verifying MOS digital circuits, but it is very expensive in terms of the computing resources required, since transistor-level circuit simulators such as SPICE (Simulation Program with Integrated Circuit Emphasis) have proven to be very time consuming in terms of computer and human effort even for relatively small circuits. It is easier to verify circuit implementation at the higher level of its description (without unnecessary details) — functional or logical level.

The step to raise the level of circuit description is performed by decompiling transistor circuit. As with the decompilation of programs, the circuit is decompiled to replace its representation at a low (transistor) level with a higher-level representation (at the logic gate level). Tools for the recognition of high-level structures in transistor circuits can be used to support many tasks in integrated circuit design, such as functional verification [1], fault simulation [2], automatic test pattern generation [3], circuit reengineering [4], static timing analysis, etc.

In the paper, we consider the problem of extraction of logical networks from transistor-level circuit netlists in SPICE. The most general case is considered when the source cell library is unknown. Graphs are used to represent both the flat transistor circuit and hierarchical network of logic elements. This is undirected vertex-colored (or labeled) sparse graph of large size. In graph interpretation, the problem is formulated as recognition of subgraphs corresponding to logic gates and other subgraphs that frequently occur in a given graph. The complexity of frequent subgraph mining was thought to be tremendous due to the need to solve the subgraph isomorphism problem, which is NP-complete, many times. But VLSI transistor netlists tend to be sparse enough and have the specific structure, so runtimes did not grow unreasonably since sensible data structures and data processing methods have been adopted.

Graph methods are proposed to solve some key tasks in the problem of decompiling transistor-level circuits. The presented graph methods have been implemented in C++ as a part of a decompilation program, which was tested using practical transistor-level circuits. The presented experimental results show that the typical running time for large CMOS circuits is polynomial in the total number of transistors in netlists.

2. Related work

There were many attempts to solve the problem of extracting the hierarchy of large-scale subcircuits from a transistor level networks for various VLSI technologies, restrictions, solution methods. An overview of the approaches to this problem can be found in [5, 6]. Some of the methods for extracting logical networks are based on structure recognition and use a rule-based method in which CMOS gate structures are recognized in transistor-level circuits as channel connected sequences of MOS transistors [5, 7, 8]. Such algorithms are very fast and can easily find static CMOS logic gates, but cannot help to recognize other structures.

The other approach [9, 10] for solving subcircuit extraction problem is based on mapping transistor-level circuit into a graph and treating subcircuit pattern matching problem as subgraph isomorphism one. However, such algorithms are much slower than rule-based techniques because the subgraph matching problem is NP-complete in general case.

Some of the methods suppose that cell library is predefined. So the subgraphs to be found are known and the problem can be reduced to pattern recognition. This is done in [9, 10] and in the second step of extraction process for the gate-level structure in [5].

This paper presents the methods and the computer program for extracting the hierarchy of a large-scale digital circuit from its MOS transistor-level description for the most general case when any predefined cell library of logic gates is unknown. Moreover, the proposed method makes it possible to recognize subcircuits that implement the same logic functions but are not topologically isomorphic at the transistor level. The method is based on the solution of graph problems that are modified to process large transistor-level descriptions in a short time.

3. Transistor circuit graph representation

The source and resulting circuit netlists are presented in SPICE (Simulation Program with Integrated Circuit Emphasis) format [2]. It is one of the main formats for exchanging electrical circuits that allows to describe both the transistor- and gate-level circuits including hierarchical ones. This format is used in the developed decompilation program for source and resulting netlists.

The main part of the circuit netlist in SPICE format is the list of transistors, where each transistor terminal is indicated by the label of the net connecting it with the rest of the circuit. Each transistor has four terminals: drain, gate, source, and substrate, and so it has four connections. The general form of the netlist description of a unipolar transistor is as follows:

$$M\langle name\rangle\langle nd\rangle\langle ng\rangle\langle ns\rangle\langle nb\rangle\langle\text{model-name}\rangle[L=\text{value}]\ [W=\text{value}],$$

where M is the title of a transistor; nd, ng, ns, and nb are the labels of nets connected with the drain, gate, source, and substrate terminals of the corresponding transistor; “model-name” is the transistor type; L and W are the length and the width. For example, the transistor instance description «mp 2 1 3 3 P» is an abbreviated notation for the pairs (mp.d, 2), (mp.g, 1), (mp.s, 3), (mp.b, 3), where the name mp of the p-MOS transistor is taken out, the names of its terminals are omitted and set by a predetermined sequence of nets.

A simple model for an electrical circuit is a hypergraph, in which the vertices correspond to devices, and edges to their connections. But in a netlist format, an electrical circuit consists of elements that are connected to each other by nets, and a more convenient natural way to represent circuits is to use an undirected bipartite graph $G = (V_1, V_2, E)$, $V_1 \cap V_2 = \varnothing$, were vertices may be divided into two classes V_1 and V_2. The vertices of the first set V_1 correspond to transistor terminals and circuit ports (primary inputs and outputs), and the vertices of the other set V_2 correspond to connections between the terminals, i.e., nets. No edge exists between two transistor terminals and no edge exists between two nets. Examples of nets are power supply and ground nets, which are connected to a large number of circuit elements. Each edge $e \in E$ has one end in V_1 and the other in V_2.

Circuit representation in the form of a hypergraph requires significantly more memory during software implementation than the representation in the form of a bipartite graph. We can say that according to the memory requirement, the complexity of the first structure is estimated as $O(n^2)$, while the complexity of the second is estimated as $O(n)$, where n is the number of circuit elements. In addition, a bipartite graph is a natural formal model for representing a circuit in SPICE format.

For sparse graphs (such as our bipartite graph), the optimal data structure that speeds up computation is an array of adjacency lists of graph vertices. The array is indexed by vertices, and each vertex of the graph corresponds to an adjacency list consisting of vertices adjacent to it. In a bipartite graph, which is a model of a CMOS circuit, the degrees of all vertices in the set V_1 of transistor terminals (and circuit ports) are 1, so for this graph each of the adjacency lists consists of the only element. In this case, the array is indexed by the vertices exclusively of the set V_1. The value of the i-th array element is the net connected to the i-th terminal. For example, the data structure representing a bipartite graph for the inverter circuit:

```
.subckt inverter 1 2 3
mp 2 1 3 3 mypmos
mn 2 1 0 0 mynmos
.ends
```

will be the array “2 1 3 3 2 1 0 0”. The order of the nets in the array of adjacency is determined by the order of the transistor instances in the SPICE circuit description. The memory requirements for such structure are estimated as $O(n)$, where $n = |V_1|$.

4. Definitions and notation

As stated above, transistor circuits are modeled as a bipartite graph $G = (V_1, V_2, E)$, consisting of two subsets of vertices V_1 and V_2 corresponding to terminals (transistor terminals and circuit ports) and nets, and the set of edges E. We assume that the graph is undirected and vertex-colored, i.e., each vertex has a color associated with it, that is drawn from a predefined set of vertex colors $L(V)$. Each vertex of the graph is not required to have a unique color, and the same color can be assigned to several vertices in the same graph. Transistor-level circuits made by CMOS technology have several types of their nodes: terminals (drain, gate, source and substrate) of n-MOS and p-MOS transistors, power supply terminals (Vdd and Gnd), input/output ports (external nets), and internal nets. Thus each graph vertex corresponding to an n-MOS terminal is assigned by one of the first four colors, p-MOS terminal is assigned by one of next four colors. Then input/output ports, Vdd and Gnd nets, internal nets have unique colors.

A graph, which is a model for describing a MOS circuit, is connected (there is a path between any pair of vertices in the graph). Some other specific features of the bipartite graph: it is sparse and the degrees of all vertices from the set V_1 of transistor terminals and circuit ports are 1.

Two undirected graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $|V_1| = |V_2|$ are isomorphic if they are topologically identical to each other, that is, there is a one-to-one mapping f between vertices of V_1 and V_2 such that each edge in E_1 is mapped into a single edge in E_2 and vice versa, i.e., $(v, u) \in E_1 \iff (f(v), f(u)) \in E_2$.

In the case of colored graphs, the mapping f must also preserve the colors on the vertices. Two bipartite colored graphs $G^1 = (V_1^1, V_2^1, E^1)$ and $G^2 = (V_1^2, V_2^2, E^2)$ are isomorphic if there is a one-to-one mapping $f : V_1^1 \leftrightarrow V_1^2$ and $V_2^1 \leftrightarrow V_2^2$ between vertices of graphs such that $L(v) = L(f(v))$ for each $v \in V_1^1 \cup V_2^1$ and each edge in E^1 is mapped into a single edge in E^2 and vice versa, i.e., $(v, u) \in E^1 \iff (f(v), f(u)) \in E^2$.

The given graph $G_s = (V_s, E_s)$ is a subgraph of $G = (V, E)$ if $V_s \subseteq V$ and $E_s \subseteq E$. Two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ of a graph G are called edge-disjoint if they do not share edges, i.e., they use different sets of edges from E: $E_1 \cap E_2 = \varnothing$. In the case of vertex-colored graphs, this mapping must also preserve the colors on the vertices.

Given two graphs $G = (V, E)$ and $G_1 = (V_1, E_1)$, the problem of subgraph isomorphism is to find an isomorphism between $G_1 = (V_1, E_1)$ and a subgraph of $G = (V, E)$, i.e., determine whether or not $G_1 = (V_1, E_1)$ is included in G. The subgraph isomorphism detection can be defined as follows: given a graph G and a pattern graph G_1 (that can have all its vertices and edges in G), find all the subgraphs of G which are isomorphic to G_1. Subgraph isomorphism has a wide range of practical applications.

Thus, graph isomorphism requires a strict correspondence among the two graphs being matched, and subgraph isomorphism requires an isomorphism between one of the compared graphs and a subgraph of the other. Subgraph isomorphism is more common than strict isomorphism in pattern recognition, but has been shown to be NP-complete for general case of graphs. However, for the problem of graph isomorphism, no efficient (polynomial) algorithm (suitable for arbitrary graphs) is known too, a lot of work has appeared on this topic, but little progress has been made [11].

5. Graph-based formulation of subcircuit extraction problem

The proposed subcircuit extraction application begins with the construction of a graph model from the SPICE description and hierarchical hash tables for storing the syntax elements of the analyzed circuit [12]. After this, the circuit is preprocessed, during which some standard fragments are searched. For example, each group of identical MOS transistors (with the same signals applied to their gate terminals), connected in series or in parallel, is replaced in the circuit with the single such transistor. Then, the identification of pass gates is fulfilled.

The goal of transistor circuit decompilation is to build a logic network that is functionally equivalent to it. The task consists in recognizing subcircuits, which implement logic gates. When there is no predefined cell library, it is necessary to extract subcircuits realizing logic functions or, if we cannot, to split the transistor circuit into sufficiently large subcircuits that look like as logic gates and are found quite often. In graph interpretation, the problem is solved by partitioning a graph into sufficiently large edge-disjoint subgraphs in such a way that they can be partitioned into the minimum number of classes of isomorphic graphs.

In MOS transistor circuit, not every subcircuit is correct. Correct subcircuits are among channel connected sequences of transistors. Thus, first, the proposed method of the subcircuit recognition uses the structural approach to divide the transistor-level circuit into subcircuits, which are channel connected sequences of transistors, as in [5, 7, 8]. In graph interpretation, the task is reduced to searching for connectivity components in a graph.

After this step, we get a set of possible correct subcircuits, which potentially can be standard CMOS gates. And, in a general case, in addition to the set of channel connected components of transistors, individual transistors or some other elements can remain. The set of such circuit elements forms the uncovered part of the circuit, they are no longer analyzed and are included in the resulting mixed gate-transistor-level circuit without changes.

In the second step, we have the set of possible subcircuits that are channel connected components. Among these subcircuits there are those that are standard CMOS gates. The task is to find such subcircuits and the functions that they realize. And finally, the set of all subcircuits, both implementing and not implementing CMOS gates, is partitioned into classes of topologically identical. Subcircuits of the same class represent the same functional block in resulting hierarchical description of the decompiled circuit. In graph interpretation, the task is to classify subcircuits into classes of isomorphic circuits.

As result of the mentioned steps performing, a hierarchical mixed gate-block-transistor netlist is generated. In the next step, the extraction of logic network from the hierarchical transistor-level circuit is done. That makes it possible to recognize more complex elements than gates. In graph interpretation, the task is to extract out of undirected graph connected subgraphs only with those vertices that correspond to CMOS gates, and convert the resulting undirected subgraphs into oriented ones.

6. Partitioning a graph into connected subgraphs

A static MOS circuit has a well-defined structure that allows it to be splitted into smaller subcircuits, each of which is a group of channel connected transistors. Such a circuit component consists of transistors connected by their source and drain terminals and provides a signal path between the power Vdd and Gnd terminals. A group of channel connected transistors is a cluster with three types of external connections:

– the cluster inputs are fed only to the transistor gates;
– the cluster outputs are connected only to the transistor gates of the other clusters;
– there are connections to the Vdd and Gnd terminals.

Figure 1 shows the example of grouping transistors into two channel connected components.

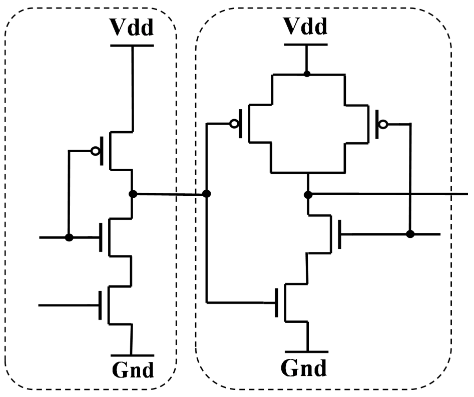

Fig. 1. Two groups of channel connected components of MOS transistor circuit

The task of recognizing clusters of the MOS transistor circuit is solved on a graph H, which is obtained from the previously introduced graph $G = (V_1, V_2, E)$, by:

– removing the circuit power terminals and transistor gate terminals;
– shorting the drain and source terminals for each transistor.

In graph interpretation, a channel connected group of MOS transistor circuit corresponds to a connectivity component of the graph H. All connectivity components are edge-disjoint subgraphs of the graph H. So the splitting transistor circuit into disjoint subcircuits of channel connected transistors is reduced to the search for connectivity components of the graph H. This is done by using the well-known Depth-First Search (DFS) algorithm, which starts at an arbitrary unconsidered vertex and explores paths from it as far as possible along each branch before backtracking. Reaching a backtracking results in a new connectivity component. When implementing the DFS algorithm, the initial graph $G = (V_1, V_2, E)$ was not transformed explicitly into the graph H. Instead, the DFS algorithm was tuned to the modification of data structure for storing a bipartite graph G.

It is worth noting that in order to search for groups of transistors connected by a current, it is necessary to know in advance which terminals of the initial transistor-level circuit correspond to Gnd and Vdd.

7. Structural recognition of logic gates

In static CMOS circuits, the MOS transistor can be regarded as a switch controlled by input voltage at its gate. The simplest digital circuit is a pass gate consisting of the only MOS transistor that controls the transmission of binary signals. This circuit is passive because it does not amplify the input signal. The amplification of binary signals is provided by a complementary MOS circuit (CMOS gate) in which, at any instant of time, gate output is connected either to a power circuit or to ground through a path with low resistance. The CMOS gate consists of two blocks separated by a connection net (output net) (Fig. 2). The first block is formed by n-MOS transistors (pull-down network — n-part of a CMOS gate), which are connected in series by their source/drain terminals. The block is placed between the connection node (output net) and Gnd. The second block is formed by p-MOS transistors (pull-up network — p-part of a CMOS gate), which are connected in parallel, by their source/drain terminals. The block is placed between Vdd and the connection node. When the block conductivities are complementary, no matter what the input signals (on transistors gates) are, there is a valid path to output node either from Gnd or from Vdd.

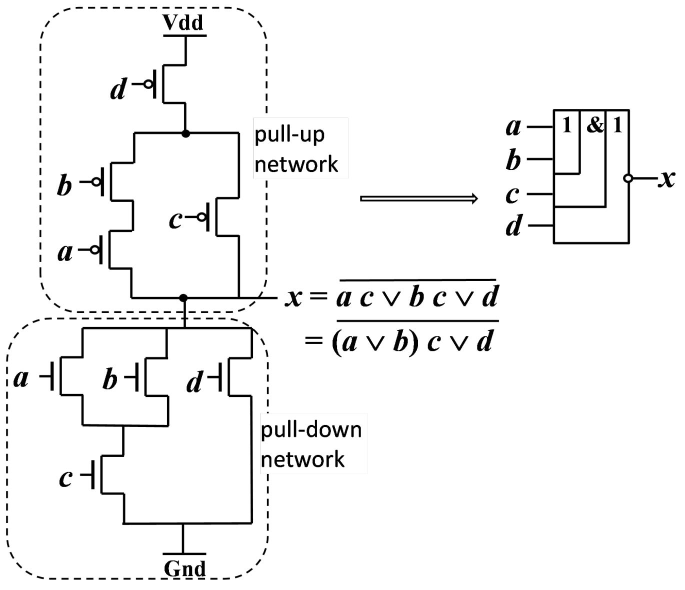
Vdd
d
pull-up network
b
c
a
x = $\overline{a\,c \vee b\,c \vee d}$
= $\overline{(a \vee b)\,c \vee d}$
a
b
d
c
pull-down network
Gnd
a
b
c
d
1
&
1
x

Fig. 2. CMOS gate: its transistor structure and implemented function

A CMOS gate is a group of channel connected transistors; the opposite is not always true. The necessary conditions for the group to belong to the class of CMOS gates are the following:

- the only chain connecting the p-part and the n-part of the group is the output (connection) node;
- all paths from the connection node go to the signal nets Gnd or Vdd;
- pull-down and pull-up networks have the same number of transistors;
- pull-down and pull-up networks implement mutually inverse functions.

For example, the right group of channel connected transistors of the two shown in Fig. 1 is CMOS NAND element, but the left one is not.

A logic function expression implemented by the pull-down (or pull-up) network is formed by tracing paths from the connection node to Gnd (or Vdd) terminal. Each path gives a conjunction of the conductivity variables fed to gate terminals of the transistors from the path. The OR of all such conjunctions yields the disjunctive normal form (DNF) for the expression. If the conductivity functions f_n and f_p of pull-down or pull-up networks are complementary ($f_n = \bar{f}_p$), then the analyzed channel connected group is a standard CMOS gate. To classify CMOS gates extracted from the transistor circuit, it is convenient to represent the recognized functions as parenthesized algebraic expressions. Such a form can be constructed by the algebraic factoring DNF of the Boolean function found [12, 13]. For the CMOS gate in Fig. 2, we have

$$f_n = ac \vee bc \vee d = (a \vee b)c \vee d, \quad f_p = \bar{a}\bar{b}\bar{d} \vee \bar{c}\bar{d} = (\bar{a}\bar{b} \vee \bar{c})\bar{d},$$

and $f_n = \bar{f}_p$. Thus, it is standard NOAO2 CMOS gate.

Channel connected groups of transistors, which are static CMOS gates, can be divided into classes of identical according to the formulae of implemented logic functions. Each class is made up of all instances that implement the same function formulae and therefore are functionally equivalent.

However, not always the only CMOS gate may be associated with the class of functionally equivalent channel connected groups. This is true if we are only interested in functional equivalence of circuits. The topological aspect requires dividing a class of functionally equivalent CMOS gates into subclasses of topologically equivalent CMOS gates. Some features of the topological implementation of circuits at the transistor level, which must be taken into account when combining or not combining subcircuits into a class of topologically equivalent, are given in [12]. For example, we should take into account the following specifics of the topological implementation of CMOS gates:

- asymmetry of the inputs of the topological implementation of a CMOS gate (although the gate implements a symmetric function);
- interchangeability of its drain and source.

The proposed algorithm distinguishes between the following groups of functionally equivalent CMOS gates:

1) CMOS gate implementations that differ from each other by exchange the drain and the source at least in one transistor. The interchangeability of the drain and source in a MOS transistor results in existence of topologically different subcircuits that implement the same logic function. For instance, there are four variants of subcircuits for a CMOS inverter. If, in a decompiled circuit, all variants of a logic gate subcircuit are represented by the same subcircuit, then the decompiled and original circuits will not be isomorphic.
2) CMOS gate implementations that differ from each other by permutation of their inputs (Fig. 3). Even if a CMOS gate implements a symmetric Boolean function, the permutation of the inputs of the CMOS circuit that implements it makes the circuit topologically nonisomorphic to the original one. This is because CMOS circuit has asymmetric inputs. However, logically, both CMOS circuits implementations are equivalent.

Topological equivalence of CMOS gate implementations can be established by checking whether the corresponding graphs are isomorphic or not.

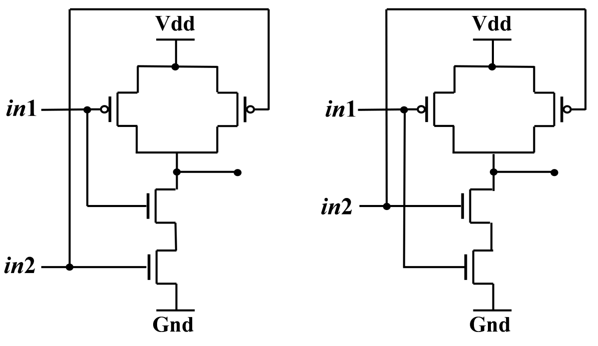

Fig. 3. Topologically nonequivalent implementations of the NAND gate

8. Graph isomorphism and canonical labeling

One of the key operations required to partition the set of subgraphs into classes of isomorphic ones consists in checking whether two subgraphs are identical or not. One way of performing this check is to perform a graph isomorphism operation. But in our case, when many such checks are required for the same set of subgraphs, a better way to perform the task is vertex canonical labeling [14]. It assigns to each graph a unique code (a sequence of bits) that is invariant on the ordering graph vertices and edges. Comparing whether or not two graphs have identical canonical labels allows you to say whether or not two graphs are identical. Moreover, by comparing the canonical labels we can partition the set of graphs into classes of pairwise isomorphic graphs. Thus, checking two arbitrary graphs for isomorphism is reduced to comparison of their canonical forms.

Calculating canonical labels is computationally equivalent to determining isomorphism between graphs; both canonical labeling and determining graph isomorphism are not known to be either in P- or in NP-complete class [15]. If a graph has $|V|$ vertices, the complexity of determining its canonical labeling using this method is in $O(|V|!)$ making it impractical even for moderate size graphs.

In our case, the complexity of determining a canonical labeling of a graph is reduced due to taking into account the special properties of subgraphs under classification: they are vertex-colored, sparse and small enough. By comparing canonical labels of graphs, it is possible to sort them in a unique and deterministic way.

Canonical labeling is done in an iterative manner in the process of building a sequence of partitions for the set of graph vertexes that defines an ordering of the graph vertices.

Suppose we have an ordered collection of subsets of the vertices $(V_1, V_2, \ldots, V_k)$ whose union is V. They say that all vertices from the same subset V_i have the same label i. The set of these subsets represents the partition of the set V of graph vertices, constructed from the initial partition that is specified by colors of vertices.

At first, the number and sizes of these subsets V_i must be the same for both compared graphs, i.e., the graphs have identical partitions of the set V. Then we repeatedly apply a relabeling step, which assigns to each vertex v a classifier $C(v) = (n_1, n_2, \ldots, n_k)$, where n_i is the number of vertices in subset V_i that are adjacent to v. Using these classifiers, each subset V_i can be partitioned into subsets, where each subset should include all vertices with the same classifier. These subsets are lexicographically ordered according to their classifiers. In this way, we may obtain a refinement of the original partition, which consists of subdividing the partition blocks. No refinement will be obtained if all vertices in each subset V_i have identical classifiers. If a refinement has been obtained, then the classifiers are recalculated (and vertices are relabeled) until there is no further refinement.

It is clear from the description that the essential idea is to relabel vertices so that each new classifier reflects information about a gradually increasing region around the vertex. In an ideal situation, after exhaustive application of the relabeling process, all subsets in the partition $(V_1, V_2, \ldots, V_k)$ will become singletons (containing exactly one member), such a graph canonical labeling is called discrete. Discrete canonical labeling defines graph canonical isomorph, which is given by an ordered set of vertex classifiers that represent a unique graph code. If two compared graphs have the same canonical labeling, then they are isomorphic with each other.

Today there exist several successful programs of computing the canonical isomorphs, they differ by refinement procedure associated with details of reducing the search tree built in the process of partition refinement (the pioneer work [16]). However, the fastest known algorithm for graph isomorphism (as well as graph canonization) is $2^{O(\sqrt{n \log n})}$ time, and no polynomial algorithm is known.

9. Graph-based subcircuit recognition method

After structural recognition of logic gates and pass gates, there are two main unrecognized groups of transistors. The first group includes structures that cannot be partitioned into gates or that are separate transistors. In resulting SPICE description they are given as ungrouped transistors. The second group includes found channel connected components of MOS transistors that have not been recognized as standard CMOS gates, so they are assigned to be pseudogates. Each of the pseudogates is represented by a bipartite undirected vertex-colored graph, which is sparse.

At this stage, the subcircuits associated with the pseudogates must be pattern matched using a user-defined library of cells, as was done, for example, in the Frosty program [5]. However, in our case, when there is no cell library, all we can do is to classify remaining pseudogates into classes of pairwise identical subcircuits.

In graph interpretation, the task consists in testing isomorphism between graphs by means of comparing their canonical labelings. To simplify the canonization problem, the subcircuit graphs are complemented with edges connecting all four terminals for each transistor. The "bliss" program (T. Yuntilla and P. Caski [17]) was chosen as a prototype of a program for calculating canonical isomorphs, which provides fast processing of large and sparse graphs. Our pseudogate graphs are represented with exactly such graphs. The experiments with the modified canonical graph labeling program have shown that applying the canonicalization process to pseudogate graphs results in a discrete canonical labeling.

The graphs of pseudogates with the same initial partitions on the set of colored vertices are considered one by one. For each of them, a canonical isomorph is generated and a hash of the canonized graph is computed. The hash value is a word-length bit string obtained by the transformation of a sequence of numbers representing graph vertex classifiers. Graphs with equal hashes are isomorphic and they are changed in a hierarchical SPICE description with their canonical isomorph.

10. Logic network construction

At this stage, we have a mixed circuit which, in addition to static CMOS gates, consists of pseudogates, pass transistors and ungrouped transistors. Now, the task is to recognize more complex elements than gates. Using the previously described graph-based subcircuit recognition method, we can recognize in logical network (consisting of CMOS gates) some library-defined patterns. So the next step is to extract a subcircuit from the mixed circuit which consists only of gates, i.e., logical network. To specify a logical network means to specify its inputs and outputs, the structure of connections between its elements, and Boolean functions realized by the elements.

In graph interpretation, a logical network is a directed connected graph $H = (W, A)$. The set of vertices W is partitioned into three subsets: network inputs and outputs, and internal vertices. Each vertex is labeled with input or output variable, or, if it is internal vertex, with the function realized by the corresponding gate. An arc (directed edge) $a =$ $= (u, v) \in A$ goes from the source vertex u to the target vertex v $(u, v \in W)$. We further consider that graph $H = (W, A)$ is specified by the adjacency list, i.e., an array D of the length $|W|$, where each entry $D[i]$ is a pointer to a linked list of all out-neighbors of vertex $w_i \in W$.

The connected graph $H = (W, A)$ is extracted from the undirected bipartite graph $G = (V_1, V_2, E)$ corresponding to the object mixed circuit. Graph H is contained in G as the connected component C, including only the vertices corresponding to the CMOS gates. There can be more than one such a component in the graph G. Each undirected connected subgraph corresponding to a connected component in a bipartite undirected graph G is transformed into a directed connected graph $H_i = (W_i, A_i)$ of some logical network. The transformation is carried out in the process of traversing the subgraph along the paths in-going or out-going from the vertices labeled as CMOS gates.

The search for the next connected component C begins with any unconsidered vertex labeled as a CMOS gate and is done by the breadth-first search (BFS) method, considering only the vertices labeled as CMOS gates. BFS allows not only to find out a connected component C, but also to get its topological sorting, which orders the vertices so that the order corresponds to reachability. That is, if a vertex u is directly reachable from v, then the edge $(u, v) \in E$ generates arc $(v, u) \in A$, and if the vertex v belongs to the i-th graph rank, then the vertex u belongs to the $(i + 1)$-th rank.

The proposed method provides to extract logic network that is ranked lexicographically. From a lexicographically ordered network of logical gates, it is easy to pass to the formulas of logical equations that specify the output functions of the network.

The next task connected with the logic network extraction is to determine primary inputs and outputs of the network. It is solved by considering fan-ins and fan-outs for all vertices of the graph $H = (W, A)$. If all vertices from both fan-in and fan-out of some vertex $v \in W$ are labeled as CMOS gates, then the vertex v is an internal one. Non-internal vertices are assigned to primary inputs or primary outputs, depending on which of the fan-in and fan-out sets contains the non-internal vertex.

After the gate-level networks are extracted, more complex elements than gates can be recognized if the cell library is known.

11. Experimental results

Some experiments with the developed decompilation program have been performed. Decompiled transistor-level circuits implement digital devices, both combinational and sequential, with the complexity of several hundred thousand transistors.

The experiments have been carried out on a computer with Intel(R) Core(TM) i5-4460 3.20 GHz and 16 GB RAM. Table 1 shows how quickly the decompilation speed decreases with increasing transistor circuit complexity. Here, transistor circuit decompilation speed is estimated by the number of its transistors processed per second: n/t, where n is the number of transistors in a decompiled circuit, t is the circuit decompilation time.

Two types of experiments have been carried out with the developed programm. In the first experiment, CMOS circuits obtained by CAD system were used. In this case, the technology cell library was known. One hundred percent coverage of the transistor-level circuit by logic gates has been obtained. In the second experiment, transistor-level circuits extracted from layouts have been examined. For some of these circuits the hierarchical SPICE models were known, for others there was no additional information other than the transistor-level circuit. In some circuits, in addition to MOS transistors, there were other primitive elements (bipolar transistors, RC elements, etc.). Here, the coverage of the transistor-level circuit with logic gates at the level of 60–70 % was observed.

Table 1

Speed of transistor circuits decompilation

Number of transistors	Decompilation time: seconds	Number of found gates	Processing speed: transistors per second
1593	0.047	570	33893
11935	0.332	2727	35948
12566	0.398	3163	31572
38356	2.603	6179	14735
52408	6.085	9091	8612
62380	6.648	13664	9380
206896	90.182	34153	2294
345301	187.151	60033	1845

Some intermediate results of applying the proposed graph methods in the subcircuit extraction program are given in Table 2. The table shows how many:

- n-MOS and p-MOS transistors are contained in each decompiled circuit (the second column);
- the numbers of found pass gates (the third column);
- the numbers of all found CMOS gates in the circuit, the numbers of functionally and topologically identical CMOS gates (the forth column);
- the numbers of all found pseudogates and the numbers of classes containing topologically identical pseudogates (the fifth column).

Table 2

Intermediate experimental results

Circuit	Number of MOS transistors	Number of pass gates	Number of CMOS gates	Number of pseudogates
1	1682, 1269	0	528, 16, 16	154, 55
2	3016, 2381	89	1041, 15, 39	284, 88
3	5776, 5827	25	2392, 7, 8	615, 23
4	5962, 5947	661	2777, 17, 34	119, 71
5	9415, 9415	1374	6639, 16, 44	0, 0
6	22988, 16436	766	5915, 39, 64	1178, 252

12. Conclusion

In this paper, we present the graph-based methods for solving the task of extracting gate-level circuits from transistor-level descriptions for the most general case when any predefined cell library of logic gates is unknown. We have used well-known graph methods, modifying them so that they process large data of special types in a short time. The proposed methods have been implemented in C++ as a part of a decompilation program. The program has been tested using practical and automatically designed transistor-level circuits. The tested circuits had up to 100000 transistors. Some results of experiments on the program execution and verification of the correctness of decompilation results can be found in [18].

Our future work is to extend the decompilation program with means of recognition memory elements in gate-level network without using pattern matching techniques.

REFERENCES

1. *Abadir M. S. and Ferguson J.* An Improved Layout Verification Algorithm (LAVA). Proc. EDAC, Glasgow, UK, 1990, pp. 391-395.
2. *Baker R.* CMOS Circuit Design, Layout, and Simulation. Third Ed. John Wiley & Sons, 2010.
3. *Kundu S.* A transistor to gate level model extractor for simulation, automatic test pattern generation and verification. Proc. Int. Test Conf. IEEE, Washington, 1998, pp. 372–381.
4. *Hunt V. D.* Reengineering: Leveraging the Power of Integrated Product Development. Vermont, Oliver Wight Publ., 1993. 282 p.
5. *Yang L. and Shi C-J. R.* FROSTY: A program for fast extraction of high-level structural representation from circuit description for industrial CMOS circuits. Integr. VLSI J., 2006, vol. 39, no. 4, pp. 311–339.
6. *Zhang N., Wunsch D. C., and Harary F.* The subcircuit extraction problem. IEEE Potentials, 2003, vol. 22, no. 3, pp. 22–25.
7. `http://www.silvaco.com/content/appNotes/iccad/2-003_LogicGates.pdf` — Logic Gate Recognition in Guardian LVS, Silvaco, 2009.
8. *Lester A., Bazargan-Sabet P., and Greiner A.* YAGLE, a second generation functional abstractor for CMOS VLSI circuits. Proc. ICM'98, Monastir, Tunisia, 1998, pp. 265–268.
9. *Ebeling E.* GeminiII: A second generation layout validation program. Proc. ICCAD-89, Santa Clara, CA, USA, 1988, pp. 322–325.
10. *Ohlrich M., Ebeling C., Ginting E., and Sather L.* SubGemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. Proc. 30th ACM/IEEE Design Automation Conf., Dallas, TX, USA, 1993, pp. 31–37.
11. *Conte D., Foggia P., Sansone C., and Vento M.* Thirty years of graph matching in pattern recognition. Int. J. Pattern Recognit. Artif. Intell., 2004, vol. 18, pp. 265–298.
12. *Cheremisinov D. I. and Cheremisinova L. D.* Extracting a logic gate network from a transistor-level CMOS circuit. Russian Microelectronics, 2019, vol. 48, no. 3, pp. 187–196.
13. *Cheremisinova L. D.* Sintez i optimizatsiya kombinatsionnykh struktur SBIS [Synthesis and Optimization of Combinational Structures of VLSI]. UIIP NAS Belarus Publ., Minsk, 2005. (in Russian)
14. *Hartke S. G. and Radcliffe A. J.* McKay's Canonical Graph Labeling Algorithm. `https://api.semanticscholar.org/CorpusID:6454900`, 2008.
15. *Garey M. R. and Johnson D. S.* Computers and Intractability: A Guide to the Theory of NP-Completeness. NY, W. H. Freeman Publ., 1979.
16. *McKay B. D.* Practical graph isomorphism. Congressus Numerantium, 1981, vol. 30, pp. 45–87.
17. *Junttila T. and Kaski P.* Engineering an efficient canonical labeling tool for large and sparse graphs. Proc. ALENEX, New Orleans, LA, 2007, pp. 135–149.
18. *Cheremisinov D. and Cheremisinova L.* Subcircuit pattern recognition in transistor level circuits. Pattern Recognit. Image Anal., 2020, vol. 30, pp. 160–169.