UDDP - UNIVERSAL DECISION DIAGRAM PACKAGE

Suzana STOJKOVIĆ
Faculty of Electronics, Beogradska 14, 18000 Nis, Serbia and Montenegro,
E-mail: suza@elfak.ni.ac.yu

SUMMARY

Decision diagrams (DDs) are frequently used and efficient data structures for discrete functions representation and manipulation. For different applications, different types of DDs have been defined. Most DD packages developed previously, such as: CUDD, BuDDy, TUD DD, BXD, CAL,... manipulate with binary decision diagrams (BDDs). This paper presents an approach for developing of a Universal Decision Diagram Package (UDDP) - application provided for manipulating with different types of shared multi-valued decision diagrams. The main idea is to develop core of a system that permits construction of various decision diagrams for different classes of discrete functions, and involve methods for DD manipulation independently on a concrete DD type. The core is simply adaptable for manipulation with different DD types. In the present version UDDP maintains manipulation with DDs for representation of functions defined in finite fields (qDDs), Multi-terminal decision diagrams (MTDDs) and Edge-valued decision diagrams (EVDDs). On the other side, UDDP is an "open source" project. It can be easy extended to the manipulation with other kinds of DDs. UDDP provides an effective component for visual representation of DDs. This component is also independent on the type of the DDs.

Keywords: *decision diagrams, programming of decision diagrams, visualization of decision diagrams*

1. INTRODUCTION

Many problems in digital logic design, artificial intelligence, telecommunications, etc. are based on manipulations with discrete functions. Because of that, various methods for representation of discrete functions have been developed. In 1986, Bryant proposed application of a graph-based representation of Boolean functions called Binary Decision Diagrams (BDDs) [2]. Analogously, for the multi-valued discrete functions representation the Multiple-place Decision Diagrams (MDDs) are defined. Recently, decision diagrams (DDs) are a widely used way for representing discrete functions with a large number of variables. Decision diagrams that represent several discrete functions are called Shared DD [2]. In the last years, many algorithms for DD manipulation have been developed [17], [18] and different kinds of DDs have been proposed to represent different classes of discrete functions [3], [13], [19].

DD programming is also a very frequently discussed problem in last years. In the papers [1] and [2] basic principles for DD programming are defined. In this paper we will discuss several DD packages, including CUDD [16], PUMA [7], BuDDy, TUD DD [8], CAL [14], etc. All these DD packages are developed on the base of the principles proposed in [1] and [2]. These packages are efficient in dealing with some particular type of binary DDs or for some classes of discrete functions. Because of that, it is a very interesting and useful task to develop a DD package for manipulation with different kinds of DDs, which could be able to realize a new kind of DDs and the corresponding manipulation algorithms.

This paper presents an approach to the programming of decision diagrams, which results in a DD package that manipulates with different kinds of DDs in a uniform way. To provide this, it is needed to develop a package core, which has to be independent of the domain and range of the function represented as well as decomposition rules applied at the nodes of the decision diagram. This core has to be adaptable and extensible to manipulation with different kinds of DDs.

This paper presents the UDDP (Universal Decision Diagram Package) - application, which is developed based on the presented approach. In the present version, UDDP is specialized for working with multi-valued qDDs (DDs that represent the functions defined in finite fields), MTDDs (Multi-terminal DDs) and EVDDs (Edge-valued DDs). UDDP contains a component for visual representation of DDs that is independent of the kind of the DD. The object-oriented technology is used in both design and implementation of the UUDP. It enables an easy outbuilding of the package for manipulation with other kinds of DDs. For the object-oriented design of UDDP, we are using UML and RationalRose tool. The package is implemented in MS Visual C++.

2. DECISION DIAGRAMS

Decision diagrams are acyclic directed graphs that contain non-terminal nodes, terminal nodes and edges. In a DD for a function f with q-valued variables, non-terminal nodes are labeled with variables x_i in f and have q outgoing edges. Outgoing edges are labeled with all possible values for a variable x_i. Terminal nodes contain the values of the function f at the points defined by n-tuples, which label edges from the root node to the corresponding terminal nodes.

DDs are classified with respect to the number of outgoing edges of non-terminal nodes, type (logic, integer, or rational numbers) and range of values for terminal nodes and with respect to the processing that is possible to be done. For example:

- BDDs are DDs where non-terminal nodes have two outgoing edges – they represent Boolean functions;
- MDDs are DDs where non-terminal nodes have more than two outgoing edges – they represent multi-valued discrete functions;
- FDDs (functional decision diagrams) are DDs where terminal nodes contain the Reed-Muller spectral coefficients;
- MTDDs are DDs in which terminal nodes values can be of any type and range;
- EVDDs are DDs in which edges contain additive data assigned as weights of the edges.
- qDDs are DDs that represent discrete functions, which are defined in finite fields, i.e. qDDs are DDs where values for a terminal node are in a finite set {0,1, …, q-1}.

Example 1. Fig. 1 shows BDD of a Boolean function $f(x_1,x_2,x_3)$ defined by the truth-vector F=[0,0,0,0,0,1,1,1] and a MTMDD of a ternary function $g(x_1,x_2,x_3)$ defined by the truth-vector G= [0,0,0,0,0,0,0,0,0,0,5,3,0,5,3,0,5,3,0,5,3,3,3,3,3,-2,-2].

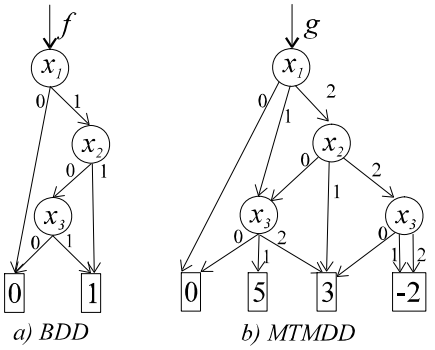

Fig. 1 BDD for the function f (a) and MTMDD for the function g (b) in Example 1

3. RELATED WORKS

3.1. Programming of DDs

Previous experiences in DD programming show that the basic problems in DD package implementation are to:

- choose an appropriate data structure for representation of nodes in the DD;
- support basic principles for DD programming;
- select an efficient algorithm for generation of DD.

This section presents an approach to the solution of the specified problems, which is used in existent DD packages.

3.1.1. Data structures for representation of a node in the DD

Data structure for BDD node representation, defined in [2] which is usually used in existing BDD packages is shown on Fig. 2. In the structure `high` and `low` note pointers to the successor nodes, `index` – the level of the node in BDD, `id` – unique node identification number, `value` – the value of terminal node, `ref_counter` – number of input edges and `mark` notes if the node is processed in some manipulation algorithm with the BDD.

```
struct node
{
  node *high,*low;
  int index;
  int value;
  int id;
  int ref_counter;
  boolean mark;
}
```

Fig. 2 Data structure for BDD node representation

The similar data structure for representation of nodes in a MDD suggested in[11] is shown in Fig. 3.

```
typedef struct node *DDedge;
typedef struct node
{
   int ref;
   char value,flag;
   DDedge next, previous;
   DDedge edge[0];
 } node;
```

Fig. 3 Data structure for MDD node representation

3.1.2. Basic principles for DD programming

Basic DD programming principles are defined in [1] and [2]. Most of the existing DD packages are implemented by their use. The principles propose to:

1. Support dealing with shared DDs. – In shared DDs, some nodes in the graph is shared by more functions. The number of nodes in shared DDs is smaller than the sum of nodes in separate DDs.
2. Store nodes into a unique node table. – Using the unique node table guarantees that at any time there are no isomorphic subgraphs and redundant nodes. To minimize time for searching a node in the table, the node table is usually realized as a hash table.
3. Support strong canonicity. – Due to the existence of the unique table, two equivalent functions are represented by exactly the same subgraph within the shared DD. This property is referred as strong canonicity.
4. Have a unique compute table – Compute table keep few recently computed functions. This table is also implemented as a hash table.
5. Use complemented edges – If edge pointed to a subgraph representing considered function is denoted as complemented then complemented values of function are used. Using of complemented edges is one of the ways to reduce the size of a DD defined as the number of nodes in the DD [2].

6. Perform an efficient memory management. - In DD manipulation, a large number of DDs are constructed and than deleted. Nodes, which are no longer used, are not freed immediately. Instead, a garbage collection is called from time to time to recover all the unused memory.
7. Support a dynamic variable reordering. - DD size depends on the order of variables. Dynamic variable reordering implies that every time when the number of non-terminal nodes grows up to a limited size, the re-ordering process is invoked automatically.

3.1.3. DD building and manipulation

There are different ways for representation of discrete functions (truth-table, cubes…). Due to, different algorithms for DD building have been developed. They are all based on building of partial DDs and contain series of operations on DDs. In the case of qDDs, operations in finite fields are used. In other DDs considered in this paper, operations corresponding to the type of values of terminal nodes (integer, real, complex…) are used. One of the basic principles for DD programming is usage of the unique compute table. Due to, all operations in finite field are improved by using one operator. In the binary logic, this is the ITE operator [1], but in multi-valued logic it is the CASE operator [17].

ITE operator is a 3-variable (F,G,H) Boolean function defined as: **If *F* then *G* else *H***, and in a formal way as:

$$ite(F,G,H) = F \cdot G + \overline{F} \cdot H \qquad (1)$$

CASE operator in q-valued logic is a (q+1)-variable function defined as follows:

$$CASE\left(F, G^0, G^1, \cdots, G^{q-1}\right) = G^i \text{ for } F = i\,. \qquad (2)$$

Instead of the CASE operator, the paper [12] proposes using of MIN and MAX operators.

3.2. Visualization of DDs

Problem of visualization of DDs is not enough resolved in existent DD packages. For example, several packages use external programs for drawing oriented graphs. For instance, CUDD and BuDDy use DOT program, which contains its own algorithm for placing nodes at the levels. Therefore, nodes from the same ‘natural’ level in DD, can be written in different levels in the picture. Because of that, pictures of DDs generated by DOT and similar programs are often not sufficiently descriptive.

DD package Jade [4] contains its own component for visual representation of DDs, which determine the position of nodes according to the position of their first appearance in the complete decision tree. Because of that, picture of a DD can be unsymmetrical. Fig. 4 shows an example of the pictures of DD generated by Jade.

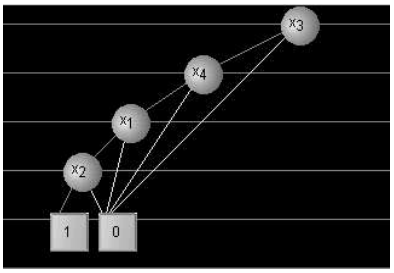

Fig. 4 Picture of a DD generated by Jade

In the package PUMA, the same arrangement of nodes is used, but the nodes are allocated equidistantly throughout the level. In this package, all aesthetic components of visual representation are hard-coded (color of the nodes and edges, size of the nodes, distance between levels, etc), and all edges are drawn as straight lines. Because of that a picture of DD can contain many edges crossing and passing of edges through the nodes. An example of picture of a DD generated by PUMA is shown in Fig. 5.

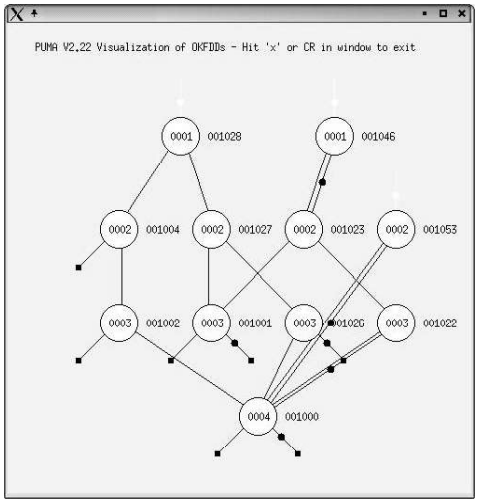

Fig. 5 Picture of DD generated by PUMA

4. DD PROGRAMMING APPROACH USED IN PROGRAMMING OF UDDP

The basic goal in UDDP programming is to provide ‘universality’, i.e. feasibility of working with different kinds of DDs. To achieve this goal, the DD package should:

- Support traditional DD programming principles (discussed in Section 3);
- Contain ‘universal methods’ (methods applicable to different types of DDs) always when it is possible;
- Replace the existing 'universal method' by a more efficient one whenever it is possible;
- Enable an easy extension of the package to manipulation with other DDs type.

'Universal methods' use 'universal' operations on DD nodes that can be implemented in different ways for concrete kinds of DDs. For example, for building of MDD, the operations MIN and MAX are used. In the case of qDDs, these operations are realized by using of CASE operator, as suggested in [12]. In the case of MTDDs MIN and MAX is realized by calling a 'universal' recursive method for evaluation of a binary operation on DD nodes. This method is introduced because algorithms for different operations on DD nodes are different only at the level of terminal nodes. A programming code for evaluation of any binary operation on DD nodes in MTDDs is shown in Fig. 6.

```
BinOp(a,b,opCode)
{
 Result = computeTable.search(
   new computeFunction(a,b,opCode));
 if ( result != 0 )
  return result;
 if ( a->level==0 && b->level==0 )
  result=(*FunctionTable[opCode])(a,b);
 else
 {
  maxLevel = max(a->level,b->level);
  for( i=0; i<a->succNo; i++ )
   succ[i]=BinOp(
     a->cofactor(maxLevel,i),
     b->cofactor(maxLevel,i),
     opCode);
  result=getNode(
    new Nonterminal(maxLevel,succ));
 }
 computeTable.add(new ComputeFunction(
   a,b,opCode,result));
 return result;
}
```

Fig. 6 Method for evaluation of binary operation on DD nodes in MTDD

A 'universal' method for evaluation of any binary operation on DD node in qDDs has been proposed in [5]. This method is based on a relationship between the definition tables of operations in finite fields and the CASE operator, which is introduced in [5]. In that paper, there are shown: a relationship between the definition table of a Boolean operator and realization of that operator by ITE and the relationship between definition table of a q-valued operator and the corresponding CASE operator.

Let Boolean operator OP is defined by a definition table V as shown in the Tab. 1.

OP	0	1
0	$v_{0,0}$	$v_{0,1}$
1	$v_{1,0}$	$v_{1,1}$

Tab. 1 Definition table of Boolean operation *OP*

If two switching functions a and b are represented by BDDs, and operator *OP* is defined by the definition table V, computation of *aOPb* can be realized by the ITE operator as follows:

$$OP(a,b) = ite(a, ite(b, v_{1,1}, v_{1,0}), ite(b, v_{0,1}, v_{0,0})) \tag{3}$$

If a q-valued operator *qOP* is defined by a definition table V as shown in the Tab. 2, and two multi-valued functions a and b are represented by qDDs, then the operator *qOP* can be realized by the CASE operator as:

$$\begin{aligned} qOP(a,b) = CASE(a, & CASE(b, v_{0,0}, \cdots, v_{0,q-1}), \\ & CASE(b, v_{1,0}, \cdots, v_{1,q-1}), \ldots, \\ & CASE(b, v_{q-1,0}, \cdots, v_{q-1,q-1})) \end{aligned} \tag{4}$$

qOP	0	1	…	*q*-1
0	$v_{0,0}$	$v_{0,1}$	…	$v_{0,q-1}$
1	$v_{1,0}$	$v_{1,1}$	…	$v_{1,q-1}$
⋮	⋮	⋮	⋯	⋮
q-1	$v_{q-1,0}$	$v_{q-1,1}$	…	$v_{q-1,q-1}$

Tab. 2 Definition table of a q-valued operation *qOP*

Besides in building, the operations on DD nodes are also used in different DD manipulations. One of often-resolved problems is calculation of a spectral transform over DDs. In [5], is shown a generic approach to calculation of spectral transforms over DDs. Spectral transform computation on DDs can be realized as a set of operations of addition (+) and multiplication (*) in the corresponding algebraic structure where the transform is defined. Because of that, a method for spectral transform calculation can be realized as a 'universal' method. This method uses the operations ADD for addition and MUL for multiplication that are realized in different ways for different kinds of DDs (similarly to the realization of MIN and MAX operations).

It is shown that an object-oriented approach in both UDDP design and implementation is the most convenient way to achieve the proposed goals. To satisfy the universality we should use parameterized classes. For example, most of defined DD manipulation algorithms are independent of the terminal node values type. Mechanism of replacing a given method with another one is available in the object-oriented approach (in the derived classes virtual methods from based classes can be re-defined). Because of that, our approach in UDDP development is: to define a set of basic classes including all universal methods for DD manipulation. For a concrete type of DDs, to define appropriate system of classes derived from defined basic classes in which, some methods from base classes can be re-defined and some new methods (characteristic for concrete DD type) can be added. In that case, addition of a package for new types of DDs representation will be simple. Classes for representation and manipulation of a new type of

DDs will be derived from the classes for representation of the most similar DD type.

5. FEATURES OF UDDP

An MDD program package should provide:

1. Manipulation with a different types of shared MDDs (where BDDs are one special case of MDDs);
2. Efficient visual representation of DDs;
3. Representation of DDs suitable for using by different modules, applications and Internet;
4. A user-friendly interface.

The basic problem in development visual representation of a decision diagram is to determine optimal arrangement of nodes (arrangement with a minimal number of edges cuttings and minimal number of edges breaks-through the nodes). Second, visual representation of DDs should not be static. It should enable a manual moving of DD elements in the viewer and showing different information about DD elements (nodes and edges).

To transfer data between different modules, applications, and through Internet, XML format is usually used. Because of that, UDDP should to contain a converter of internal DD representation into the XML format, and vice versa.

A user of UDDP package need not to be familiarized with implementation details, but he must specify a way for construction of a DD; processing that will be done; and an information about final DD which he is interested in. It follows that UDDP should contain an efficient visual user-interface for management by DD manipulation.

6. UDDP IMPLEMENTATION DETAILS

Based on the approach discussed in Section 4, we build the UDDP package. UDDP is provided for manipulation with different kinds of shared multi-valued DDs. In both design and implementation process of UDDP, the object-oriented approach (as suggested in Section 4) is used. For the design process, the UML (Rational Rose tool) is used while for a package implementation the C++ programming language is used.

For realization of four fundamental features of UDDP reviewed in Section 5, there are four basic components provided in our application: DDPLibrary, Graph, XMLProcessor and IOManager. The main diagram of UDDP architecture is shown in Fig. 7.

6.1. DDPLibrary

`DDPLibrary` is in charge for creating and different processing of DDs. `DDPLibrary` contains a set of base classes including all universal methods for DD manipulation, and classes specialized for qDDs, MTDDs (with different types of terminal nodes values) and EVDDs (with different types of terminal nodes and edges values) manipulation. Package `DDPLibrary` contains three subpackages: `Node`, `Engine` and `DDCore`.

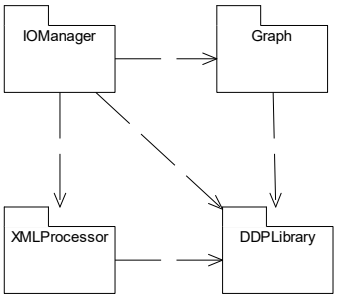

Fig. 7 Main-diagram of UDDP architecture

As shown in Section 3, the first task in DD programming is to choose efficient data structure for DD node representation. `DDPLibrary` contains class system for DD node representation grouped in the package `Node` (see Fig. 8).

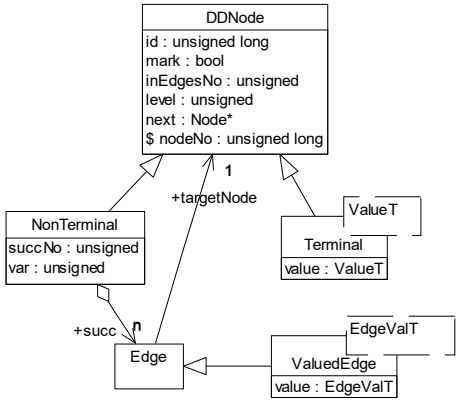

Fig. 8 Class diagram of the package Node

Base class in the system is a `DDNode` class containing common attributes of all DD nodes such as: unique id, mark, number of input edges, level and pointer of the next node at the same level. From this class, classes `Nonterminal` and `Terminal` are derived. Class `Nonterminal` contains the corresponding variable index, dynamic vector of outgoing edges and their number. In ordered DDs, there are non-terminal nodes labeled with the same variable at the same level. In that case, the class `Nonterminal` has not to contain attribute `var` (variable index). This adding of attributes enables manipulation with unordered DDs. Representation of edges as a dynamic vector enables manipulation with heterogeneous DDs (whose different non-terminal nodes have different number of outgoing edges). Class `Terminal` contains the value of the represented function. The function value can be of the different type. We use standard C++ types and type `Complex`. It can be any type with defined operators <<, >>, =, ==, <, + and *. Class `Edge` represents edges in DDs. This class contains pointer to the target node. UDDP can also manipulate with

EVDDs. In EVDDs, the class `ValuedEdge` represents an edge. This class contains value of the edge (that is also of the optimal type). If it is needed additional information about edges, a new class for edge representation can be derived from the class `ValuedEdge`.

Classes responsible for storing and manipulation of a DD node are grouped into the package `Engine`. Fig. 9 shows class diagram of the package `Engine`.

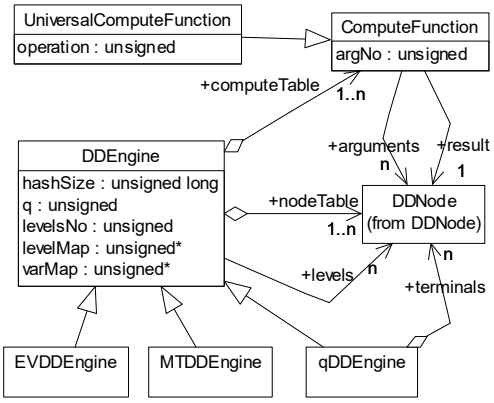

Fig. 9 Class diagram of the package Engine

Basic class in this package is a `DDEngine`. It contains both, unique node table and compute table (which are realized as hash tables). In the `DDEngine` all basic operations on DD nodes, needed for DD building and manipulation, are defined as pure virtual methods. There are implementations of these operations in the classes derived from `DDEngine` because the same operation is realized in different ways for different DD type. For example, in qDDs all DD node operations can be realized by using only one operator (CASE operator). One operator usage speeds up DD manipulation because a finding of a recent computation in the compute table is more probable. The compute table for qDDs is also very simple. It is sufficient to memorize only the input arguments and result of every executed operation. In other DD types, for each executed operation, the operation type has to be known. This problem can be resolved in two ways: by using different compute tables for different operations or by using one compute table in which for each computation the operation code is memorized. In our UDDP the second way is used.

There are set of classes for DDs representation in the package `DDCore`. Fig. 10 shows the class diagram of the package `DDCore`.

Basic class for multi-valued decision diagram representation is a `MDD` class. It contains all standard DDs manipulation methods (for building of DD on the base of different representation of discrete functions, function composition, cofactor computing, spectral transforms... All methods in this class are realized as virtual. If for a concrete type of DDs there exist efficient algorithms for some methods defined in this class, in the derived class (representing a concrete DDs type) these methods are redefined. In classes representing concrete DDs type several methods are realized for few spectral transforms. For example, in `qDD` class there exist methods for GF and RMF transforms, in `MTDD` class there are methods for Walsh and arithmetic transform calculation, etc. There is also a universal `spectralTransform` method in the `MDD` class. This method is based on an algorithm proposed in [5] and realizes any spectral transform defined by a basic transform matrix and by definition tables for operations + and *. Definition tables of operations are used only in qDD, in other type of DDs operations + and * coincide with arithmetic operations of addition and multiplication.

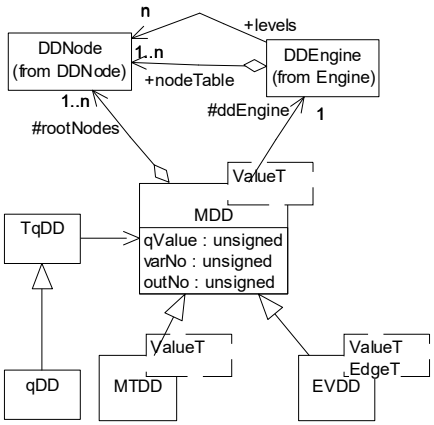

Fig. 10 Class diagram of the package DDCore

The component `DDPLibrary` is a basic component for DD representation and manipulation. Because of that, it was developed independently of other components of the system, and can be compiled and used absolutely in both DOS and UNIX (LINUX) operating systems.

6.2. Graph

Graph component is in charge for visual representation of DDs. For drawing of DDs, the UDDP uses an algorithm for drawing directed graphs proposed in paper [6]. This algorithm contains four steps:

- placing the nodes in discrete levels;
- setting the order within levels;
- setting layout coordinates of nodes;
- drawing of edges.

In DDs, nodes are assigned to the levels. Because of that, first step in DD drawing is creation of a primary visual representation of the DD where Y coordinates of nodes are determined by their levels

and X coordinates are determined by order of nodes in depth-first traversal of graph.

The goal of ordering nodes within levels is minimizing the edge crossings. This step is realized by using an iterative algorithm that is proposed in [6]. Each iteration of that algorithm consists of three actions: computing median values of nodes (median values of X coordinates of neighbor nodes), sorting of nodes by median values throughout the levels, and transposition of neighbor nodes at the levels while edge crossings are reduced. The proposed number of iterations in that step is 24.

The goal of shifting the nodes by X coordinates throughout levels is minimization of edges lengths. In our program, X coordinates of non-terminal and terminal nodes are determined in different ways. For setting of X coordinates of non-terminal nodes, an iterative algorithm based on computation of the median values is used. Terminal nodes are placed evenly from the left to the right margin of the picture.

In the last step edges are drowned as Bezier curves with four control points. Bezier curves are used when angles of edge crossings should be increased and when edges should to bypass the nodes.

Example 2. Figures 11 – 14 show pictures of the DD of the function ADD2 (2-bit adder) after each step of drawing algorithm.

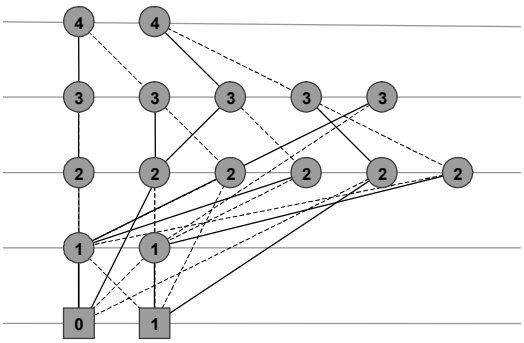

Fig. 11 The first picture of the DD of the ADD2

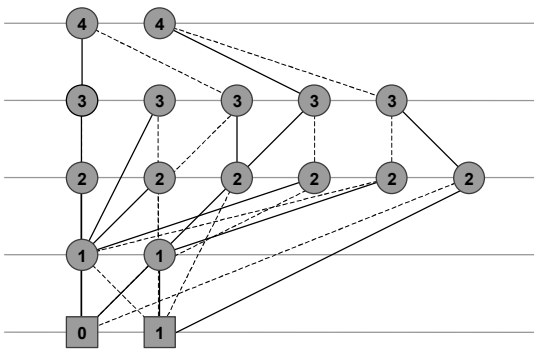

Fig. 12 Picture of the DD after node ordering

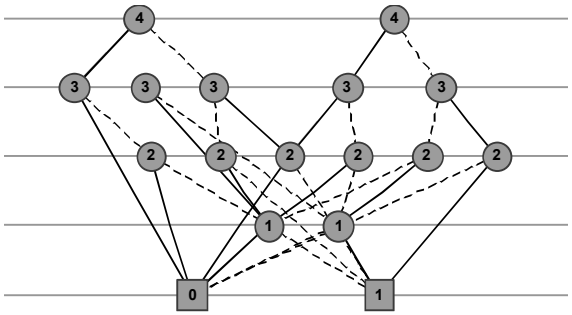

Fig. 13 Picture of DD after X coordinates setting

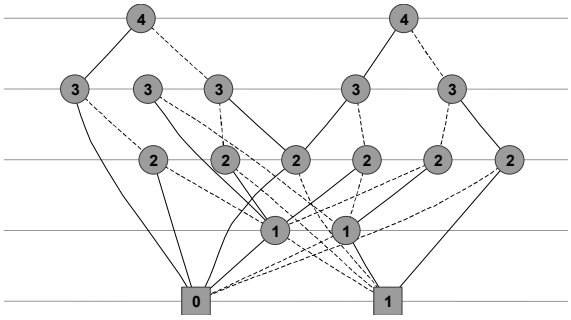

Fig. 14 Picture of DD after edges placement

Visual representation of DD in UDDP is not static. It enables manual relocation of nodes; manual deformation of edges, displaying different information about node or edge in the DD; writing a graph to a file, etc. User can define complete appearance of the graph (size and color of the nodes, distance between levels, data which will be shown at the graph, etc.). Each element of the graph can be marked by one left click. Reallocation of the marked element is limited. Nodes can be moved only at the same level; edges can be distorted, but choice and target node cannot be changed. More information about a particular node or edge is shown after left double-click on the corresponding node (edge).

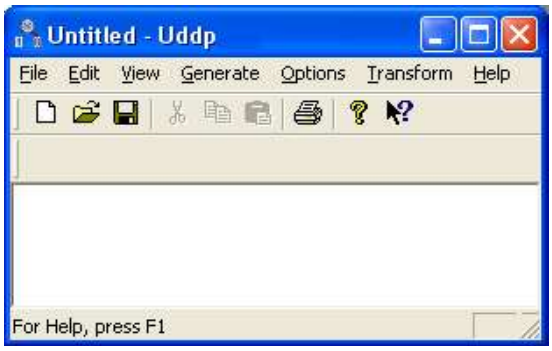

Fig. 15 The main window of the UDDP

6.3. IOManager

IOManager is in charge for the user interface. UDDP is a Windows application and IOManager enables communication between user and application by using Windows resources (menus,

dialogs, icons…). Main window of the application is shown in Fig. 5.

6.4. XMLProcessor

XMLProcessor is responsible for conversion of an internal DD representation into the XML format, and vice versa. XMLProcessor contains two components: XML-writer and XML-reader. XML-writer converts internal DD representation into XML format while XML-reader interprets a XML format (builds DD on the base of its XML file). In XMLProcesor implementation MSXML 4.0 is used.

7. CONCLUSION

DDs are a state-of-the-art data structure used in modern VLSI CAD tools. Especially, BDDs are very interesting. Several packages are provided for BDDs manipulation. In the last decade there is a renew interest in multi-valued logic. MDDs are efficient data structure for multi-valued discrete function representation and manipulation. Unlike to BDD manipulation packages, the efficient MDD manipulation packages are not developed yet. Some particular examples are given in [11].

This paper presents the Universal DD Package, which manipulates with qDDs, MTDDs and EVDDs. However, it is designed in such a way that represents an open source project and can be easily adapted to deal with any other MDD types. UDDP is a Windows application containing tools for visual representation of DDs and their conversion into the XML format.

REFERENCES

[1] Brace, K S, Rudell, R L, Bryant, R E: Efficient implementation of a BDD package, In Design Automation Conference, San Francisco, June 1991, 417-421.

[2] Bryant, R E: Graph-based algorithms for Boolean functions manipulation, IEEE Trans. on Computers, Vol. C-35, No. 8, August 1986, 677-691.

[3] Bryant, R E: Binary Decision Diagrams and Beyond: Enabling Tehnologies for Formal Verification, International Conference on Computer-Aided Design ICCAD '95, November, 1995, pp. 236-243.

[4] Drechsler, R: JADE: Implementation and Visualization of a BDD Package in Java, http://www.informatik.uni-bremen.de/grp/ag-ram/doc/work/DATE_uni_booth.pdf.

[5] Drechsler, R, Jankovic, D, Stankovic, R S: Generic implementation of DD Packeges in MVL, Proc. EUROMICRO '99, Milano, 1999, pp. 352-358

[6] Gansner, E R, Koutsofios, E , North, S C, Vo, K-P: A Technique for Drawing Directed Graphs. IEEE Transactions on Software Engineering, March 1993, pp. 214-230.

[7] Hett, A, Drechsler, R, Becker, B: The DD Package PUMA – An Online Documentation, http://ira.informatic.uni-freiburg.de/software/puma/puma.htm, 1996.

[8] Horeth S, Blank C: TUD Decision Diagram Package, Preliminary C Programmers Manual, http://www.rs.e-technik.th-darmstadt.de/~sth/dd/dd.html

[9] Janssen, G: Design of a Pointerless BDD Package, Proc. 10th Int. Workshop on Logic & Synthesis,Granlibakken, Lake Tahoe, CA,2001

[10] Miller, D M, Drechsler, R: CMVL DDs package, Proc. 28th Int. Symp. on Multiple-Valued Logic, Fukuoka, Japan, 1998, 494-512.

[11] Miller, D M, Drechsler, R: Implementing a multiple-valued decision diagram package, Proc. 28th Int. Symp. on Multiple-Valued Logic, Fukuoka, Japan, 1998, 52-57.

[12] Miller, D M, Drechsler, R: On the construction of Muliple-Valued Decision Diagrams, Proc. 32nd Int. Symp. on Multiple-Valued Logic, Boston, USA, 2002, 245-253.

[13] Minato S I: Graph-Based Representation of Discrete Functions, Chapter in Representations of Discrete Functions, edited by T. Sasao, M. Fujita

[14] Sanghavi J V, Ranjan R K, Brayton R K, Sangiovanni-Vincentelli A: High Performance BDD Package Based on Exploiting Memory Hierarchy, Proceedings of ACM/IEEE Design Automation Conference, June 1996

[15] Sasao, T, Fujita, M: Representations of Discrete Functions, Kluwer Academic Publishers, 1996.

[16] Somenzi, F: CUDD Release 1.1.1, 1996.

[17] Srinivasan A, Kam T, Malik S, Brayton R K: Algotithms for Discrete Function Manipulation, Proceedings of the International Conference on Computer-Aided Design, November 1990, 92-96.

[18] Stankovic, R S; Stankovic, M; Jankovic, D: Spectral Transforms in Switching Theory, Definitions and Calculations, Nauka, Belgrade, 1999.

[19] Stanković, R S, Sasao T: Decision Diagrams for Discrete Functions: Classification and Unified Interpretation, ASP-DAC'98, February 1998, pp. 439-446.

BIOGRAPHY

Suzana Stojković was born on 7.10.1966. in Niš, Serbia and Montenegro, and received B.Sc. and M.Sc. degrees from Department of Computer Science of Faculty of Electronic Engineering in Niš in 1990 and 1996, respectively. Since 1991 she is working as a teaching assistant at the Department of Computer Science. Hers research interest includes decision diagrams, multi-valued logic and object-oriented software design and implementation.