Binary Decision Diagram Visualization: A Research Experience for Undergraduates[*]

M. A. Thornton, R. W. Skeith, S. M. Karp, J. N. Taylor

Department of Computer Engineering
University of Arkansas
Fayetteville, AR 72701-1201
(501)575-5159 (office)
(501)575-5339 (fax)
mitch@engr.uark.edu

ABSTRACT

A Research Experience for Undergraduates (REU) project is described. A discussion of the technical aspects of the project is given as well as experiences gained by the undergraduates. The benefits of including the undergraduates in the research effort resulted in increased project productivity, encouragement in pursuing graduate work and a nontraditional classroom experience for the Junior and Senior level students akin to that derived from other capstone design projects.

The REU aspect of the research project has indicated that it is not only a valuable learning experience for the undergraduates, but an effective means to obtain tangible contributions to the research project as a whole and an effective recruiting incentive for future graduate students. Both undergraduates have already committed to staying in school to obtain the M. S. degree. Among their peers, the undergraduate research appointment is viewed as desirable and has beneficial effects on the entire undergraduate student population. The success of the REU involvement in the project has motivated the authors to always attempt to include undergraduates in research projects.

[*] This project is sponsored by NSF grant CCR-9633085

1.0 INTRODUCTION

A NSF sponsored research project has been underway at the University of Arkansas for the past 2 years. As part of this project, an REU supplement was requested and granted in August 1997 and August 1998. Since then, two undergraduate students have been employed to assist in the project. Their duties have been to aid the GRA in developing code and to design and implementation of a graphical user interface for visualizing research results. Our experience from this REU project is that the research laboratory is one of the most effective learning environments for both graduate and undergraduate students on a campus. Some specific areas where undergraduate students benefited from this project are:

- Hardware Description Language (HDL) syntax
- Familiarity with elementary parsing theory
- Pass-Transistor Logic (PTL) circuits
- Digital logic design
- Data Structures
- Netlist syntax and translation

2.0 PROJECT DESCRIPTION

The existence and use of Computer Aided Design (CAD) tools for digital circuit design is due to the recent development and improvement of techniques to efficiently represent large and complex designs. Binary Decision Diagrams (BDD) are internal computer data structures that are used to represent digital logic circuits. BDDs are similar to binary trees with the addition of shared nodes. With shared nodes, a BDD may reuse common nodes of the circuit for efficiency. BDDs are characterized by the number of outputs, inputs, levels, and graph nodes present in the structure. Circuit designers can use BDD reduction tools to reduce circuit size and complexity. In most cases, the BDD representation of a circuit is so large that it is difficult – almost impossible – to visualize the BDD. The focus of this project is to create a tool to visualize the BDD.

2.1 PROJECT IMPLEMENTATION

The goal of the project is to produce a graphical representation of a BDD for output to a display. This tool is used in support of the overall research project to produce new methods for minimizing the size of BDDs. The input circuit is defined by an ISCAS netlist (figure 3) and a BDD variable order file. The output is a Visual Compiler Graph (VCG) graphical view (figure 6) that supports zoom in, zoom out, and pan. Alternatively, a high resolution GIF, Postscript, JPG, or bitmap can be generated for a more portable output. For illustrative purposes, the C17 benchmark circuit (figure 1) will be used to describe the process.

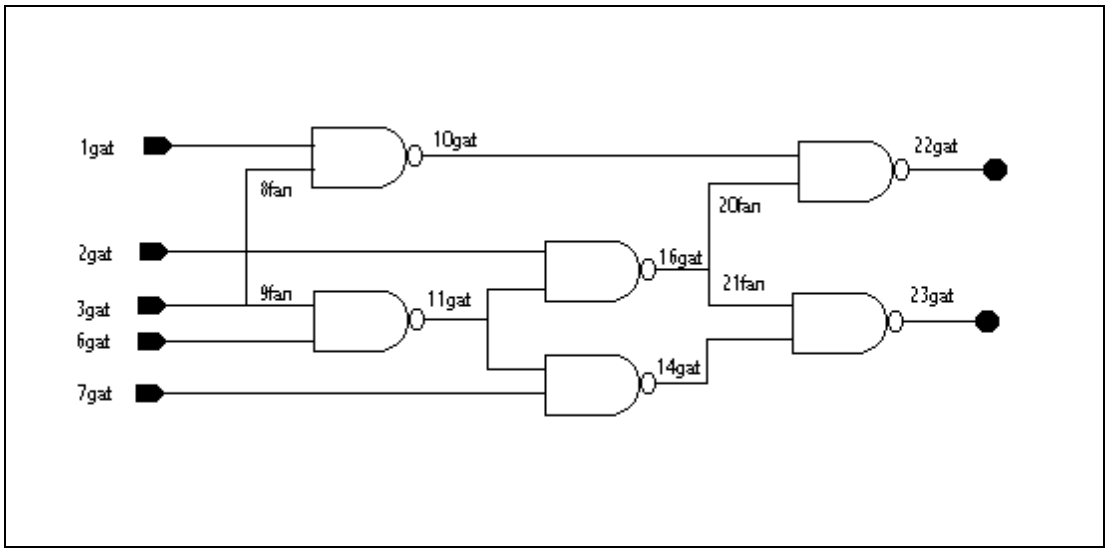

Figure 1: The logic gate representation of the C17 circuit

VCG is a high performance, graphing tool that is capable of graphing trees. Given a unique list of nodes, edges, labels, and node order, VCG creates one of two things. It can create an interactive representation of the BDD within the VCG viewer that incorporates zoom, pan, and search capabilities. Alternatively, it will output a bitmap image of the BDD with a user defined resolution and file name. The later option is useful for displaying the BDD in other applications such as web browsers and graphic viewers. As the number of nodes in the BDD increase, the user-defined resolution for the bitmap must

also increase to ensure that the zoom feature is smooth. VCG is not resource intensive and is extremely fast – drawing large BDDs in a matter of seconds.

The first step in the process is to convert the ISCAS (figure 3) netlist and the BDD variable order file into a usable format. Figure two contains the format of the ISCAS file.

Unique ID for each net	*Net name*	*Type*	*Number of Fan-outs*	*Number of Fan-ins*	*Extra information*

Figure 2: The format of the ISCAS file

The *net name* may be "#gat" or "#fan" where "#" is the *unique identification number* for the net. *Gat* and *fan* mean logic gate and fan in/out respectively. *Type* is the either the name of the logic gate (ie. NAND) or "from" if the net is a fan in/out from another net.

```
*c17 iscas example
*-------------------------------------------------
    1     1gat inpt    1   0       >sa1
    2     2gat inpt    1   0       >sa1
    3     3gat inpt    2   0 >sa0 >sa1
    8     8fan from     3gat       >sa1
    9     9fan from     3gat       >sa1
    6     6gat inpt    1   0       >sa1
    7     7gat inpt    1   0       >sa1
   10    10gat nand    1   2       >sa1
     1     8
   11    11gat nand    2   2 >sa0 >sa1
     9     6
   14    14fan from    11gat       >sa1
   15    15fan from    11gat       >sa1
   16    16gat nand    2   2 >sa0 >sa1
     2    14
   20    20fan from    16gat       >sa1
   21    21fan from    16gat       >sa1
   19    19gat nand    1   2       >sa1
    15     7
   22    22gat nand    0   2 >sa0 >sa1
    10    20
   23    23gat nand    0   2 >sa0 >sa1
    21    19
```

Figure 3: Sample ISCAS file for C17 circuit

The *bdd_print_bdd()* function accepts an ISCAS and an ordering and outputs a human readable format (figure 4). The human readable format describes the traversal of the BDD. All the information necessary to generate the VCG input file is contained in the human readable format. This project converts human readable format into VCG input format. The conversion is implemented in the *bdd_print_vcg()* function.

Output 1

```
if var.0
  if var.1
    1
  else if !var.1
    var.2
  endif var.1
else if !var.0
  if var.1
    if var.2
      !var.3
    else if !var.2
      0
    endif var.2
  else if !var.1
    var.2
  endif var.1
endif var.0
```

Output 2

```
 if var.0
  if var.1
    !var.2
  else if !var.1
    1
  endif var.1
else if !var.0
  if var.1
    if var.2
      0
    else if !var.2
      var.3
    endif var.2
  else if !var.1
    var.3
  endif var.1
endif var.0
```

Figure 4: Human readable format of C17

The second step is to parse the human readable format line by line. The *lex C* library provided the support needed to break each line into tokens and pass them to *bdd_print_vcg()*. The *lex* code feeds a *C* switch statement in *bdd_print_vcg()* that converts human readable format into *VCG* format. The lex parser and the switch case are the framework of *bdd_print_vcg()*. The *bdd_print_vcg()* code compiles the *VCG* input file (figure 5) from the human readable format (figure 3). *VCG* reads the *VCG* input file and generates the graphical representation of the C17 circuit (figure 6).

```
graph: { title : "Graph"

 width: 700
 height: 700
 x: 90
 y: 30

 color: aquamarine
            /***Output 1 EDGE/TARGET LIST***/
/*****************************************/
edge: { sourcename: "1.var.0.1" targetname: "1.var.1.2" }
edge: { sourcename: "1.var.1.2" targetname: "1.1.4" }
edge: { sourcename: "1.var.1.2" targetname: "1.var.2.3" }
edge: { sourcename: "1.var.2.3" targetname: "1.1.6" }
edge: { sourcename: "1.var.2.3" targetname: "1.0.5" }
edge: { sourcename: "1.var.0.1" targetname: "1.var.1.1" }
edge: { sourcename: "1.var.1.1" targetname: "1.var.2.2" }
edge: { sourcename: "1.var.2.2" targetname: "1.var.3.4" }
edge: { sourcename: "1.var.3.4" targetname: "1.0.8" }
edge: { sourcename: "1.var.3.4" targetname: "1.1.7" }
edge: { sourcename: "1.var.2.2" targetname: "1.0.3" }
edge: { sourcename: "1.var.1.1" targetname: "1.var.2.1" }
edge: { sourcename: "1.var.2.1" targetname: "1.1.2" }
edge: { sourcename: "1.var.2.1" targetname: "1.0.1" }

            /***Output 2 EDGE/TARGET LIST***/
/*****************************************/
edge: { sourcename: "2.var.0.1" targetname: "2.var.1.2" }
edge: { sourcename: "2.var.1.2" targetname: "2.var.2.4" }
edge: { sourcename: "2.var.2.4" targetname: "2.0.8" }
edge: { sourcename: "2.var.2.4" targetname: "2.1.7" }
edge: { sourcename: "2.var.1.2" targetname: "2.1.3" }
edge: { sourcename: "2.var.0.1" targetname: "2.var.1.1" }
edge: { sourcename: "2.var.1.1" targetname: "2.var.2.2" }
edge: { sourcename: "2.var.2.2" targetname: "2.0.4" }
edge: { sourcename: "2.var.2.2" targetname: "2.var.3.3" }
edge: { sourcename: "2.var.3.3" targetname: "2.1.6" }
edge: { sourcename: "2.var.3.3" targetname: "2.0.5" }
edge: { sourcename: "2.var.1.1" targetname: "2.var.3.1" }
edge: { sourcename: "2.var.3.1" targetname: "2.1.2" }
edge: { sourcename: "2.var.3.1" targetname: "2.0.1" }
```

```
            /***Output 1 NODE LIST***/
/*************************************/
node: { title: "1.var.0.1" label: "var.0" horizontal_order: 1 }
node: { title: "1.var.1.2" label: "var.1" horizontal_order: 2 }
node: { title: "1.1.4" label: "1" horizontal_order: 4 }
node: { title: "1.var.2.3" label: "var.2" horizontal_order: 3 }
node: { title: "1.1.6" label: "1" horizontal_order: 6 }
node: { title: "1.0.5" label: "0" horizontal_order: 5 }
node: { title: "1.var.1.1" label: "var.1" horizontal_order: 1 }
node: { title: "1.var.2.2" label: "var.2" horizontal_order: 2 }
node: { title: "1.var.3.4" label: "var.3" horizontal_order: 4 }
node: { title: "1.0.8" label: "0" horizontal_order: 8 }
node: { title: "1.1.7" label: "1" horizontal_order: 7 }
node: { title: "1.0.3" label: "0" horizontal_order: 3 }
node: { title: "1.var.2.1" label: "var.2" horizontal_order: 1 }
node: { title: "1.1.2" label: "1" horizontal_order: 2 }
node: { title: "1.0.1" label: "0" horizontal_order: 1 }

            /***Output 2 NODE LIST***/
/*****************************************/
node: { title: "2.var.0.1" label: "var.0" horizontal_order: 1 }
node: { title: "2.var.1.2" label: "var.1" horizontal_order: 2 }
node: { title: "2.var.2.4" label: "var.2" horizontal_order: 4 }
node: { title: "2.0.8" label: "0" horizontal_order: 8 }
node: { title: "2.1.7" label: "1" horizontal_order: 7 }
node: { title: "2.1.3" label: "1" horizontal_order: 3 }
node: { title: "2.var.1.1" label: "var.1" horizontal_order: 1 }
node: { title: "2.var.2.2" label: "var.2" horizontal_order: 2 }
node: { title: "2.0.4" label: "0" horizontal_order: 4 }
node: { title: "2.var.3.3" label: "var.3" horizontal_order: 3 }
node: { title: "2.1.6" label: "1" horizontal_order: 6 }
node: { title: "2.0.5" label: "0" horizontal_order: 5 }
node: { title: "2.var.3.1" label: "var.3" horizontal_order: 1 }
node: { title: "2.1.2" label: "1" horizontal_order: 2 }
node: { title: "2.0.1" label: "0" horizontal_order: 1 }
            /****END****/
 }
```

Figure 5: VCG input file for the C17 circuit

3.0 PROJECT RESULTS

This process requires very little memory because it only looks at one line of input at a time. The amount of disk space required is directly proportional to the number of nodes in the BDD. This process works efficiently for the C17 benchmark circuit, but currently has a problem with larger circuits that make use of sub-formula notation. Sub-formula notation from *bdd_print_bdd()* is a recursive method of traversing the BDD. It reduces the number of human readable output lines exponentially. The *lex* parser currently recognizes the recursive sub-formula tokens. The core switch statement in *bdd_print_vcg()* does not presently handle recursion. A recursive function is needed to

expand the sub-formulae. If the sub-formulae are not expanded into a non-recursive format, then *bdd_print_vcg()* will not output unique node names. Duplicate node names cause *VCG* to fail and no output graphic is created. The addition of recursion to this process is a small task compared to the lex parser and the switch case handler.

Once *bdd_print_vcg()* outputs a *VCG* acceptable format, *VCG* can immediately display it, save it for later, or output a bitmap. Bitmap files are not compressed and are not easily viewable from web pages. *ImageMagick* is a set of *GNU* graphic utilities that is capable of converting bitmaps to GIF or JPG images. With *ImageMagick* the bitmaps can be converted easily web friendly formats.

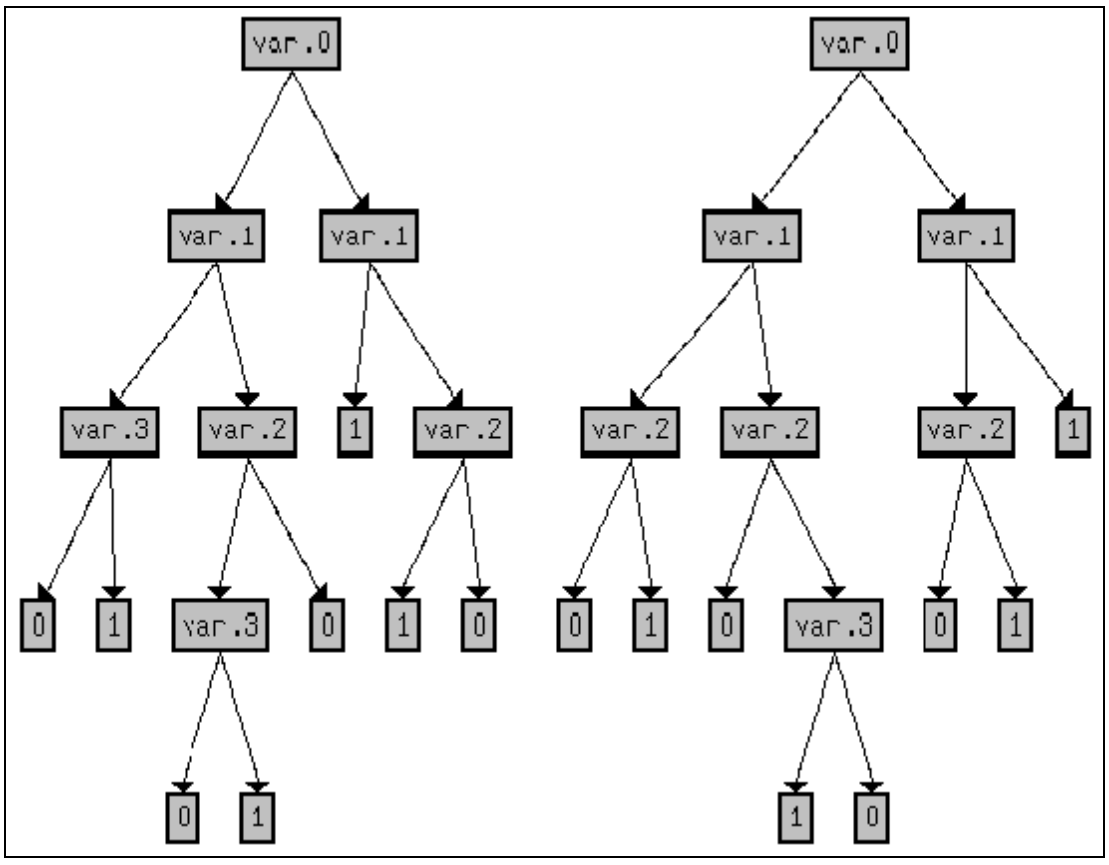

Figure 6: The VCG graphical display of the C17 circuit

3.0 CONCLUSION

An overview of a research experience for undergraduates has been given. We have described the benefits derived by the involved undergraduates as well as those achieved by the investigators. In keeping with the theme of the "non-traditional engineering classroom", our experience from this endeavor is that the academic research laboratory is a very effective setting for undergraduate education and helps to encourage the involved students to pursue graduate degrees.