PROGRAM ENCRYPTION TOOLKIT

GRAPHICAL USER INTERFACE (PETGUI)

Release 2.6

What is PETGUI?

PET graphical user interface (GUI) is a front-end to the Java-based Program Encryption Toolkit (PET).

Its primary purpose is to support visualization and analysis of information related to obfuscation and deobfuscation of digital logic circuits defined at the (netlist) gate-level.

It also provides a rich set of features for generating information about digital logic circuits themselves in terms of function, form, cryptographic properties, structure, and visualization.

Installing PETGUI

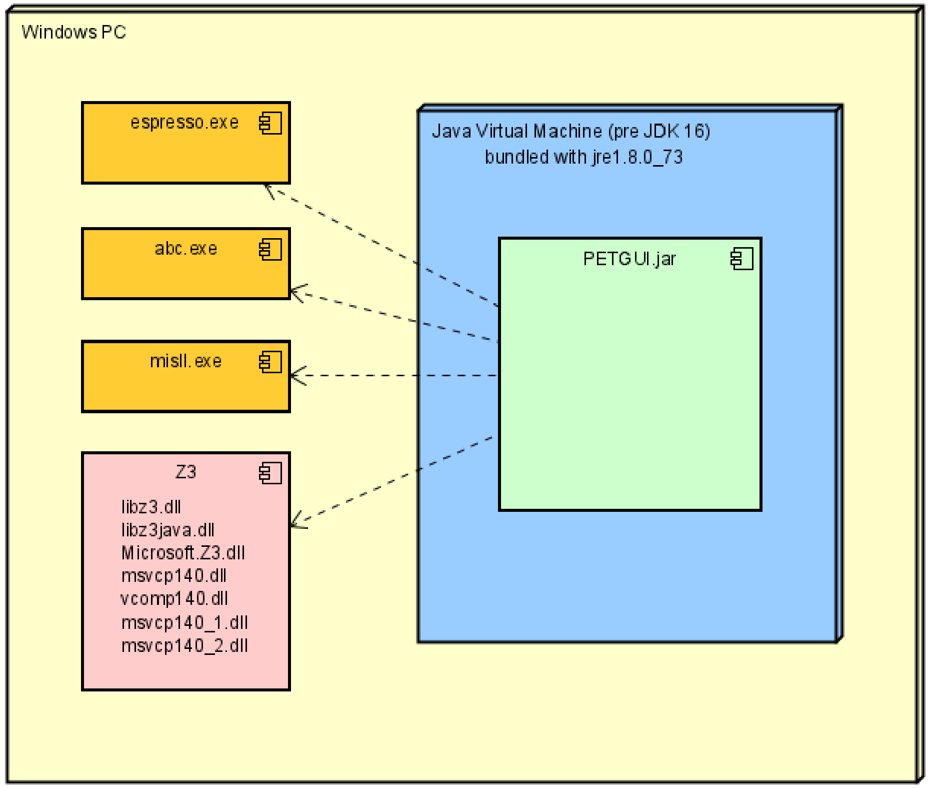
- Download and expand the entire PETGUI folder (ZIP), decompress and put it in a location of your choosing
- Put the PETGUI folder in a path that does not use spaces… this will prevent errors with some third party tools
- PETGUI has not been checked for compatibility with versions of Java below 7.X and should be compatible with any Java version below 16.X

- PETGUI is packaged as an executable JAR
  - If you have a compatible JRE installed, double-click **PETGUI.jar**
  - If you don't have a compatible JRE, PETGUI comes with a default Java runtime environment (JRE): run **PETGUI.bat**

- No other installation should be required beyond this

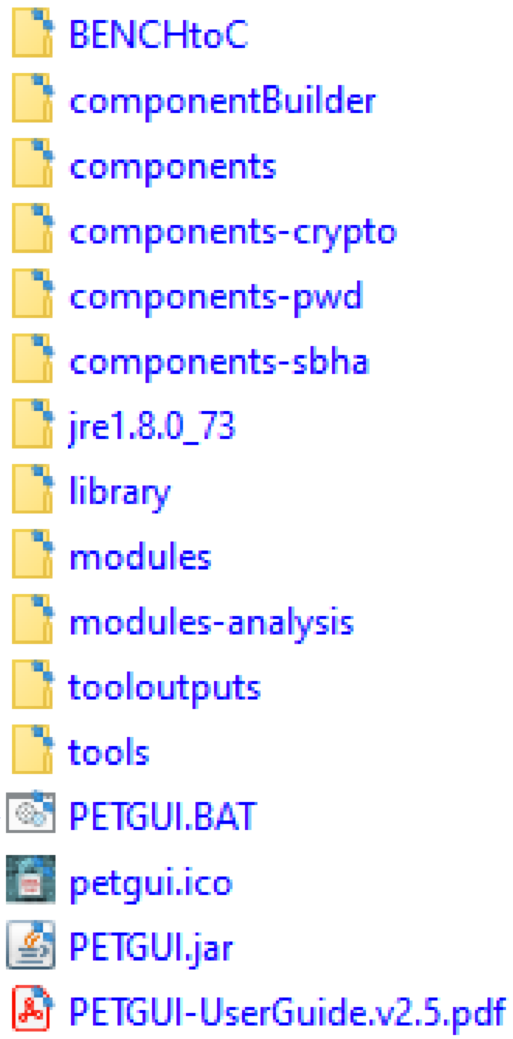
Installing PETGUI

- Although PETGUI could run under Linux or MacOS, the native tool libraries that are currently used are Windows binaries. As such, PETGUI is configured currently to run under Windows.
  - Given research interest and partnership, a version of PETGUI for LINUX can be developed and released



Running PETGUI

- For developers, the PET Javadoc API can be found in a separate ZIP download.
- A default JVM is provided, so no prior Java installation is required.
- Several folders are provided with samples, testcases, and reference component and library circuits.
- Because some tools that are used by PET are in native Windows format and there are no pure Java alternatives to them, PET is configured to run on Windows. The **tooloutputs** is used as temporary folder for several file-based operations used by Espresso, misII, and ABC.
  - This folder can be emptied on a regular basis
- PET is distributed as an executable JAR compatible with pre-Java 16 environments. A Batch file is provided to run the JRE from the provided folder.


BENCHtoC
componentBuilder
components
components-crypto
components-pwd
components-sbha
jre1.8.0_73
library
modules
modules-analysis
tooloutputs
tools
PETGUI.BAT
petgui.ico
PETGUI.jar
PETGUI-UserGuide.v2.5.pdf

Overview

- Supports basic research into adversarial analysis and obfuscation of logic circuits
- GUI provides visible functionality for research and experimentation
- Over 15 years of research
  - Master’s student research code
  - Code base underwent refactoring 2012-2016
- Provides visualization support for experiments and studies in polymorphic variation and circuit protection
- Provides basic functionality for logic circuit design and analysis

Some Research Lineage

2006
FSU

Enhanced Mobile Agent Security (McDonald)

2008
AFIT

Algorithms for White-box Obfuscation Using Randomized Subcircuit Selection and Replacement (Norman)
Obfuscation Framework Based on Functionally Equivalent Combinatorial Logic Families (James)
Software Obfuscation with Symmetric Cryptography (Lin)
Sub-Circuit Selection and Replacement Algorithms Modeled as Term Rewriting Systems (Simonaire)

2009

Characterizing Component Hiding Using Ancestral Entropy (Williams)
Removing Redundant Logic Pathways in Polymorphic Circuits (Kim)

2010

Deterministic Component Hiding Using Identification and Boundary Blurring Techniques (Parham)
Deterministic, Efficient Variation of Circuit Components to Improve Resistance to Reverse Engineering (Koranek)

1 UG Thesis
9 Masters Theses
2 Doctoral Theses

3 Grants (AFOSR, AFIT, AFRL)
5 Journal Articles
22 Conference/Workshop Papers
9 Workshops ~

2017
USA

Digital Logic Protection Using Functional Polymorphism (Forbes)

2019

Analyzing Program Protection Using Software Based Hardware Abstraction (Manikyam)
Deterministic Polymorphic Circuit Generation Using Boolean Logic Representation (Stroud)

Tool Interfaces

- PETGUI uses the following tool interfaces (see the Appendix for more information on each tool)
  - **ESPRESSO version #2.3 (native C binary) – synthesis**
  - **misII release #2.2 (native C binary) – synthesis**
  - **ABC version 1.01 (native C binary) – synthesis**
  - **JDD build 104, Feburary 2012 (fully Java) – BDD**
  - **Z3 (Java, with Windows DLL) – SAT solver**
  - **SATGraf version 0.2 (fully Java) – SAT visualization**
  - **Sat4J (fully Java) – SAT solver**
  - **yFiles v2.11.0.2 (fully Java) – graph visualization**

Native format: bench

- PET uses ISCAS BENCH format as the native format for logic circuit netlists

# = comment
Can appear anywhere
End of line or whole line

```
# 5 inputs
# 2 outputs
# 0 inverters
# 6 gates ( 6 NANDs )
```

INPUTS:
In MSB ordering
At least 1

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)
```

OUTPUTS:
In MSB ordering
At least 1

```
OUTPUT(22)
OUTPUT(23)
```

A distinguished intermediate GATE-ID

INTERMEDIATE GATES:
`GATE-ID1 = GATE-TYPE (GATE-ID2, GATE-ID3, …)`
String or Integer ID supported
Multi-fanin supported

```
10 = NAND(1, 3)
11 = NAND(3, 6)
19 = NAND(11, 7)
16 = NAND(2, 11)
22 = NAND(10, 16)
23 = NAND(16, 19)
```

Use *.bench.txt for all file BENCH file extensions

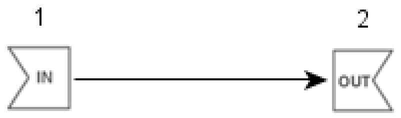
BENCH Gate Types

- Basic Types
  - INPUT
  - OUTPUT
  - AND
  - NAND
  - OR
  - NOR
  - XOR
  - NXOR (XNOR)
  - NOT
  - BUFFER (BUFF)

- Extended types
  - Constants
    - CONST0
    - CONST1
  - Sequential gates:
    - DFF
    - JKFF
    - SRFF
    - TFF

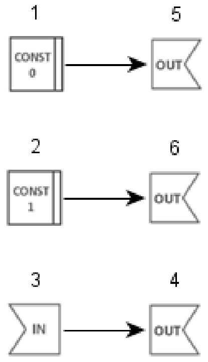
Basic Rules for BENCH Netlists

- At least 1 INPUT
- At least 1 OUTPUT



```
INPUT(1)
OUTPUT(1)
```

- No more than 1 CONST0
- No more than 1 CONST1

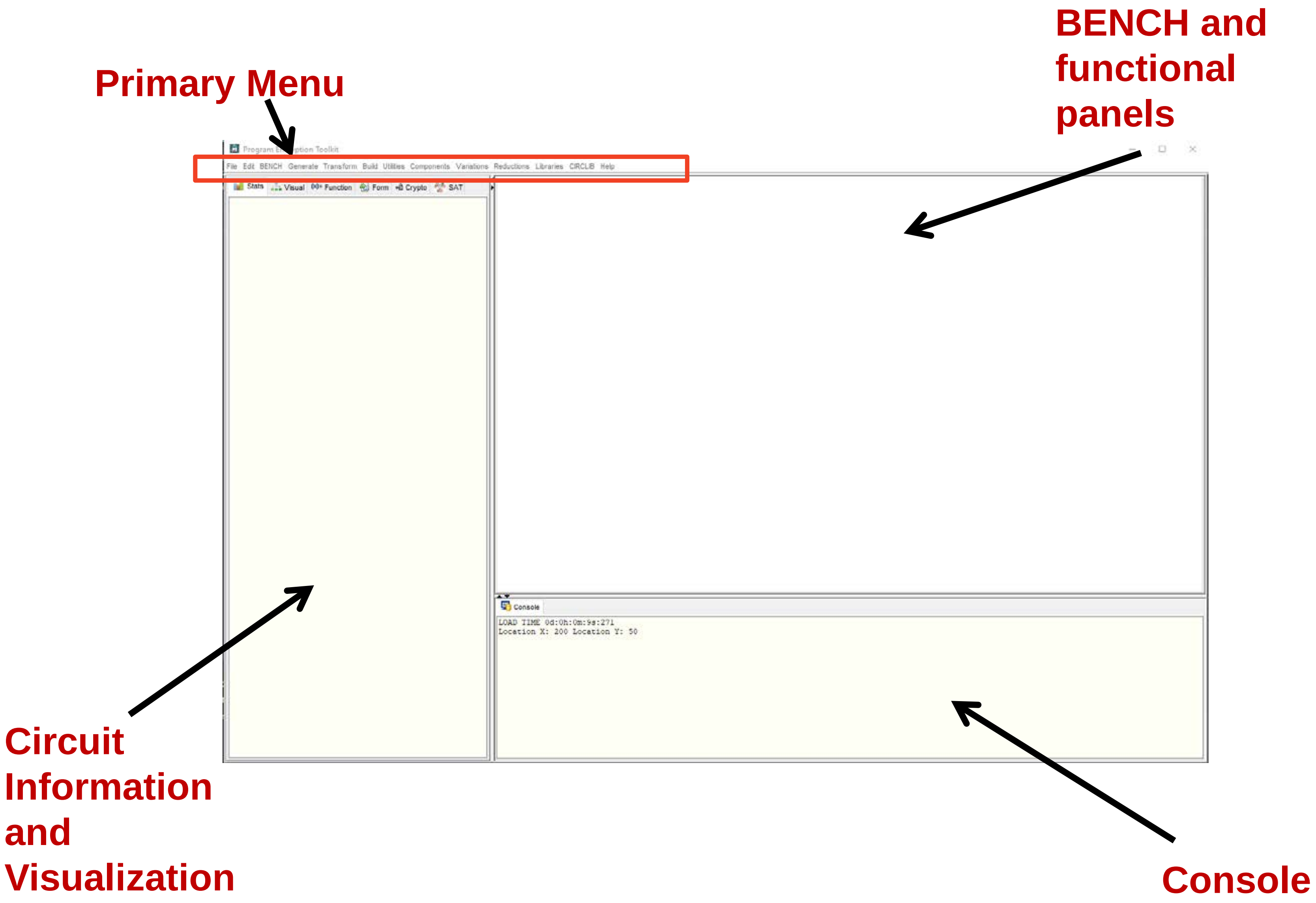


```
INPUT(3)

OUTPUT(3)
OUTPUT(1)
OUTPUT(2)

CONST1(2)
CONST0(1)
```

- *String* **gate names** are **case-sensitive**

```
INPUT(1)
input(B)

#comment
output(3)

gate1 = aND(1,B)

#comment
GATE1 = AND(1,gate1)
gATe1 = or(gate1,GATE1)
3=xnor(gate1,gATe1) # end of line comment

#comment
```

PET Graphical user interface


Primary Menu
BENCH and functional panels
Circuit Information and Visualization
Console
Program En ption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats
Visual
Function
Form
Crypto
SAT
Console

```
LOAD TIME 0d:0h:0m:9s:271
Location X: 200 Location Y: 50
```

Main Menu

Unimplemented/TBD

BENCH operations:
Compile (validate)
View Graph/Schematic/Image
Simulate
Truth/State Table and State Diagram
MROBDD/Binary Decision Diagrams
Implicants/Terms
KMAP
Canonical formulas
Functional Equivalence (TT, BDD, ABC)

Utilities:
Permutation Circuits
Random Circuit
Selections
IO Permutations
ABC Console

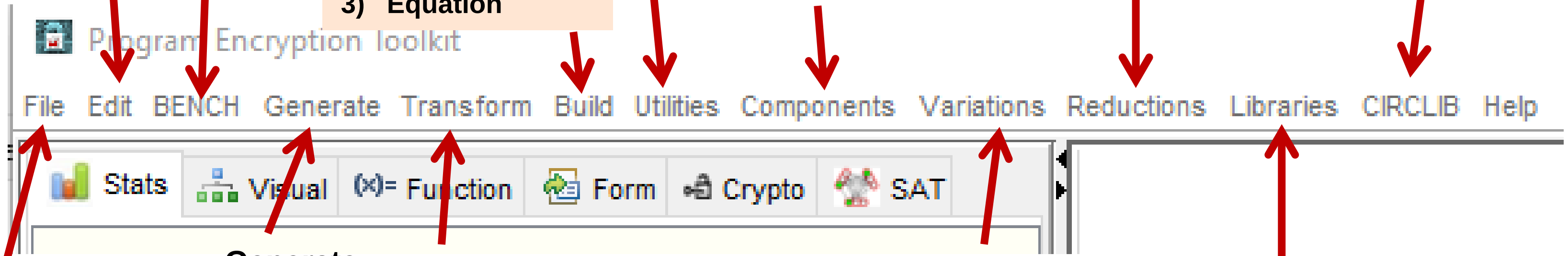
Reductions:
Equational Reducer
Circuit Reducer
Structural Pattern Reducer
Shaped Pattern Reducer
Pattern Viewer
Pattern Finder

Components:
Circuit Partitioner
Subgraph Enumeration
Semantic Component ID
Structural Component ID
Module Library

Edit:
Copy/Cut/Paste
Undo/Redo

Circuit Builders:
1) **Canvas**
2) **TT**
3) **Equation**

CIRCLIB:
Static circuit library and analysis



Generate from BENCH:
PLA
VHDL
UW
BLIF
misII
ABC
z3
DIMACS

Transforms:
Concat
Merge
Merge Common Input
Decompose mutli-fanin
Decompose XOR
Decompose function
Transform Basis
Transform SOP/POS/RSE/AIG
Transform Espresso
Transform misII
Transform ABC
Transform Tseytin

Obfuscating Variations:
Iterative Selection/Replacement
Boundary Blur
Component Fusion
Component Encryption
Program Encryption
Polymorphic Gates
Logic Encryption

Library Circuits
Basic components

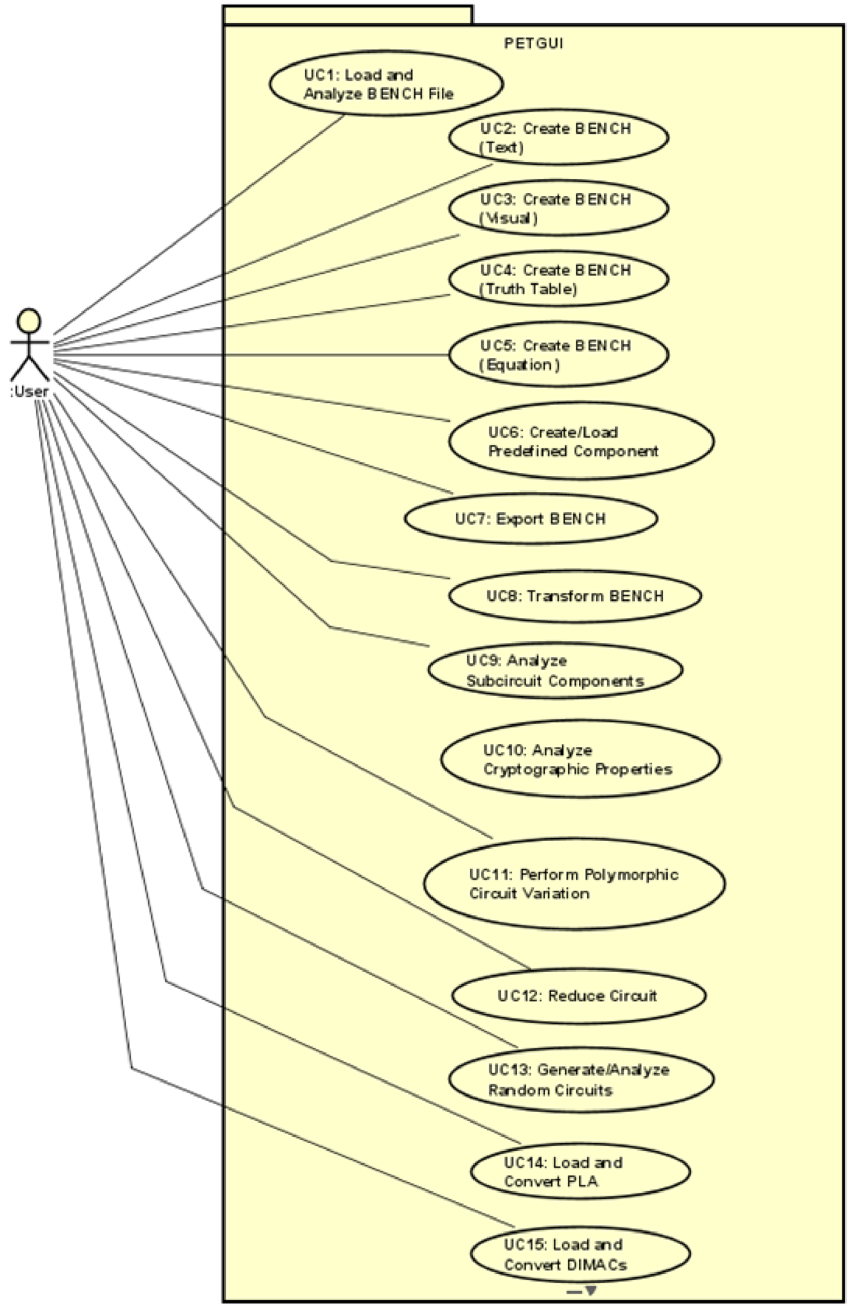
- Gates	
- Adders/Subtractors	
- Multipliers	ISCAS-85
- Decoders	ISCAS-89
- Encoders	ITC-99
- MUX/DEMUX	PLA
- Comparators	BLIF
- Polygates	

File:
New
Open:
1) BENCH
2) DIMACS
3) PLA
Close/Close All
Save/Save As/Save All
Export

Basic UI Functionality for Circuit Analysis

- Load/edit/create text BENCH files
- Compile combinational / sequential
- View circuit graph / generate circuit images
- Generate truth tables
  - Input Vectors for large input sizes
- Generate reduced minterms/PLA/BLIF formats
- Generate KMAP
- Generate structural VHDL and equational Verilog
- Generate binary decision diagrams (BDT, OBDD, ROBDD, MROBDD)
- Generate Boolean expression trees and formula
- Generate canonical standard forms
- Simulate circuit execution
- Perform cryptographic Boolean function analysis

Use Cases


PETGUI
UC1: Load and Analyze BENCH File
UC2: Create BENCH (Text)
UC3: Create BENCH (Visual)
UC4: Create BENCH (Truth Table)
UC5: Create BENCH (Equation)
UC6: Create/Load Predefined Component
UC7: Export BENCH
UC8: Transform BENCH
UC9: Analyze Subcircuit Components
UC10: Analyze Cryptographic Properties
UC11: Perform Polymorphic Circuit Variation
UC12: Reduce Circuit
UC13: Generate/Analyze Random Circuits
UC14: Load and Convert PLA
UC15: Load and Convert DIMACs
:User

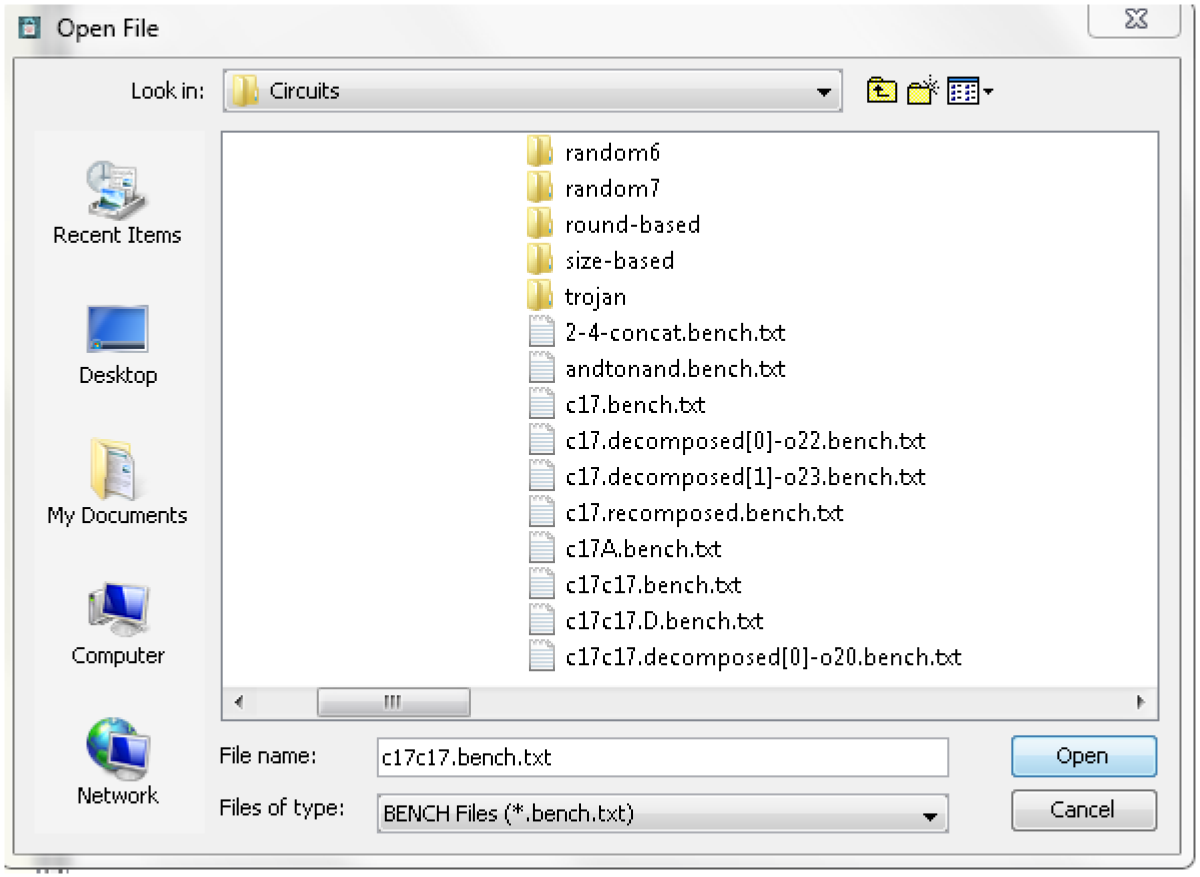
1. Load and Analyze BENCH File
2. Create New BENCH File (Text)
3. Create New BENCH File (Visual)
4. Create New BENCH File (Truth Table)
5. Create New BENCH File (Equation)
6. Create/load a Pre-defined BENCH Component
7. Export a BENCH File in Different Formats
8. Transform BENCH File into Different Forms
9. Analyze Subcircuit Component Information
10. Analyze Cryptographic Boolean Properties
11. Perform Polymorphic Circuit Transformations
12. Reduce a Circuit
13. Generate and Analyze Random Circuits
14. Load and Convert PLA File
15. Load and Convert DIMACs File

Use Case 1 Scenarios: Load and Analyze BENCH File

- Opening BENCH file
- Compiling (syntax checking)
- View Graph
- View Image
- View Schematic
- Generate truth table
- Generate implicants
- Generate binary decision diagrams (BDDs)
- Generate Boolean expression trees
- Comparing equivalence: Truth table, BDD, ABC
- Generating PLA
- Generating UW formats
- Generating VHDL
- Generating BLIF
- Generating misII information
- Generating ABC functions
- Generating z3 Model
- Generating DIMACS Model
- Generating KMAP
- Generating Normal Form equations (DNF, CNF, ANF)
- Simulating the circuit

Use Case: Loading and Analyzing BENCH File

1. File->Open->BENCH File


Open File

Look in: Circuits

Recent Items
Desktop
My Documents
Computer
Network

random6
random7
round-based
size-based
trojan
2-4-concat.bench.txt
andtonand.bench.txt
c17.bench.txt
c17.decomposed[0]-o22.bench.txt
c17.decomposed[1]-o23.bench.txt
c17.recomposed.bench.txt
c17A.bench.txt
c17c17.bench.txt
c17c17.D.bench.txt
c17c17.decomposed[0]-o20.bench.txt

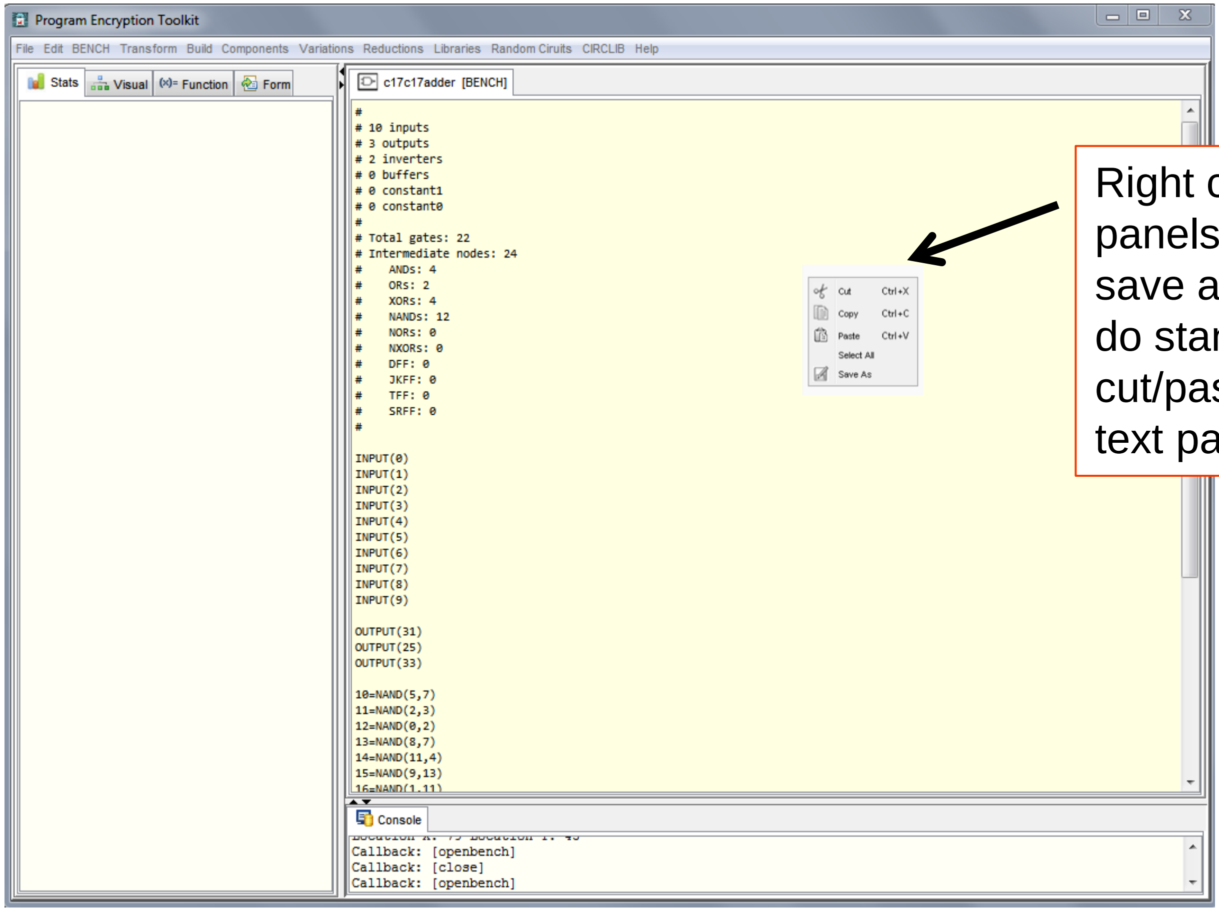
File name: c17c17.bench.txt

Files of type: BENCH Files (*.bench.txt)

Open
Cancel

Sample circuit directory included

2. BENCH->Compile Combinational (Cntrl+B)


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats
Visual
Function
Form
c17c17adder [BENCH]
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All
Save As
Console

```
#
# 10 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 24
#    ANDs: 4
#    ORs: 2
#    XORs: 4
#    NANDs: 12
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
```

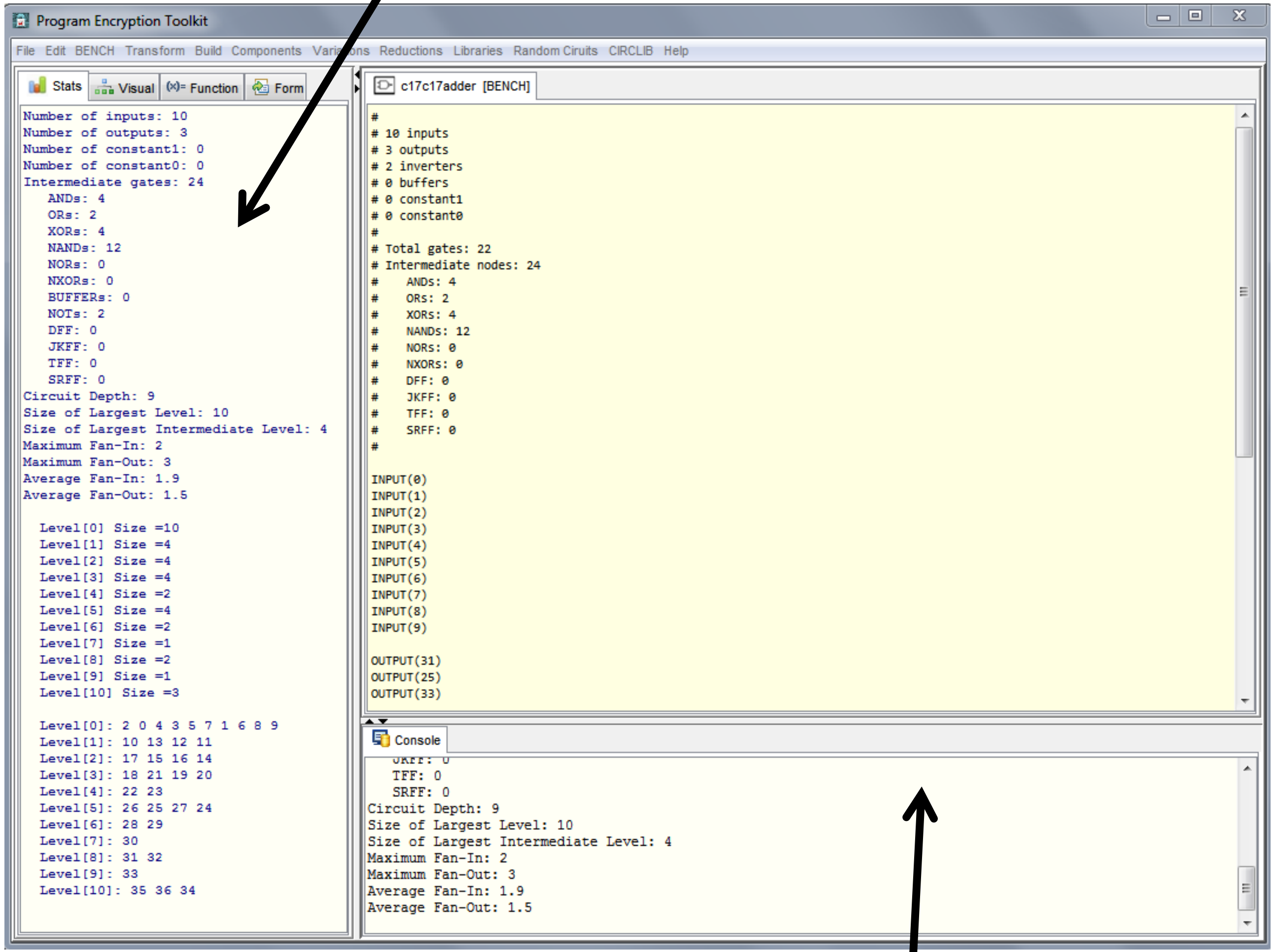
```
Callback: [openbench]
Callback: [close]
Callback: [openbench]
```

Right click for text panels: you can save any text to a file, do standard copy or cut/paste for editable text panels

You must know whether the BENCH file is sequential or not: Has loops and/or contains FF gates

Use Case: Loading and Analyzing BENCH File

On successful compile: statistics are displayed


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats
Visual
Function
Form
c17c17adder [BENCH]
Console

```
Number of inputs: 10
Number of outputs: 3
Number of constant1: 0
Number of constant0: 0
Intermediate gates: 24
   ANDs: 4
   ORs: 2
   XORs: 4
   NANDs: 12
   NORs: 0
   NXORs: 0
   BUFFERs: 0
   NOTs: 2
   DFF: 0
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 9
Size of Largest Level: 10
Size of Largest Intermediate Level: 4
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 1.9
Average Fan-Out: 1.5

  Level[0] Size =10
  Level[1] Size =4
  Level[2] Size =4
  Level[3] Size =4
  Level[4] Size =2
  Level[5] Size =4
  Level[6] Size =2
  Level[7] Size =1
  Level[8] Size =2
  Level[9] Size =1
  Level[10] Size =3

  Level[0]: 2 0 4 3 5 7 1 6 8 9
  Level[1]: 10 13 12 11
  Level[2]: 17 15 16 14
  Level[3]: 18 21 19 20
  Level[4]: 22 23
  Level[5]: 26 25 27 24
  Level[6]: 28 29
  Level[7]: 30
  Level[8]: 31 32
  Level[9]: 33
  Level[10]: 35 36 34
```

```
#
# 10 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 24
#    ANDs: 4
#    ORs: 2
#    XORs: 4
#    NANDs: 12
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)
```

```
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 9
Size of Largest Level: 10
Size of Largest Intermediate Level: 4
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 1.9
Average Fan-Out: 1.5
```

Compile errors appear in the Console

3. BENCH->View Graph

Mouse scroll wheel = zoom in/out

Graph

Visual

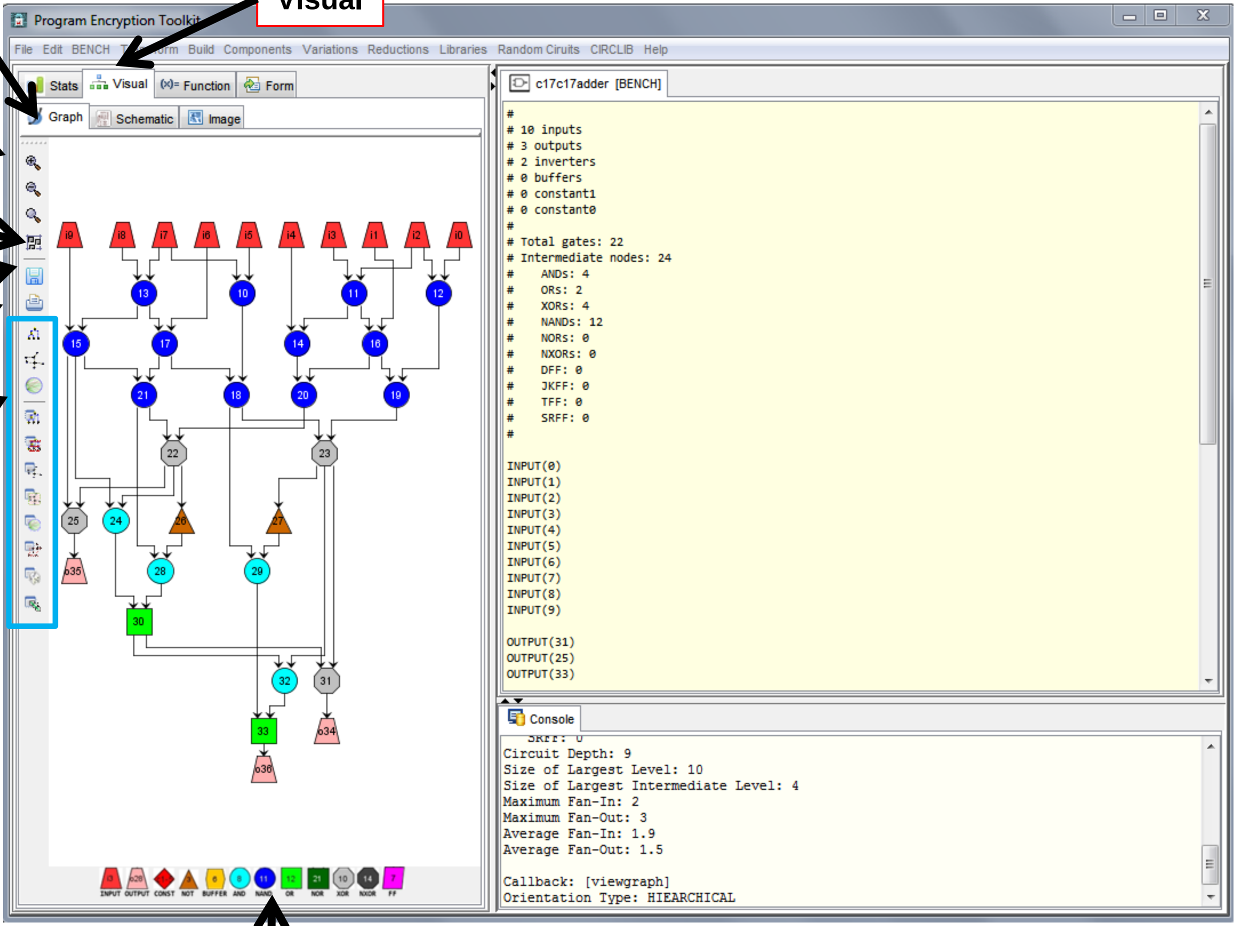
Zoom in / Zoom out

Fit+Recenter

Save Image

Print

Graph layout options


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats
Visual
Function
Form
Graph
Schematic
Image
c17c17adder [BENCH]
Console
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

```
#
# 10 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 24
#    ANDs: 4
#    ORs: 2
#    XORs: 4
#    NANDs: 12
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)
```

```
   SRFF: 0
Circuit Depth: 9
Size of Largest Level: 10
Size of Largest Intermediate Level: 4
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 1.9
Average Fan-Out: 1.5

Callback: [viewgraph]
Orientation Type: HIEARCHICAL
```

Node Gate Key

General color scheme:
Light = positive logic
Dark = negative logic

4. BENCH->View Image

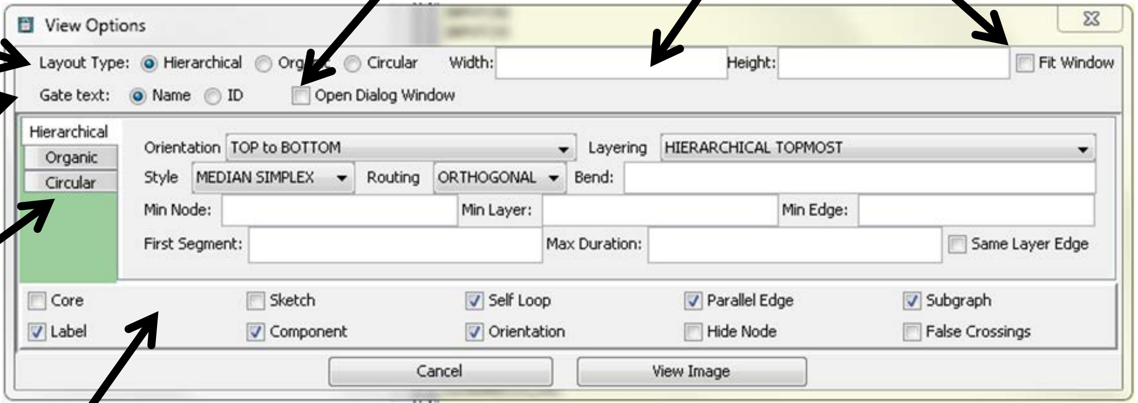
ALSO, open in a dialog window

Custom height/width OR Fit to window

Graph layout type

Show gate ID or gate Name in image

Layout options per type



View Options

Layout Type: [✓ Hierarchical] Organic Circular Width: Height: Fit Window

Gate text: [✓ Name] ID Open Dialog Window

Hierarchical / Organic / Circular

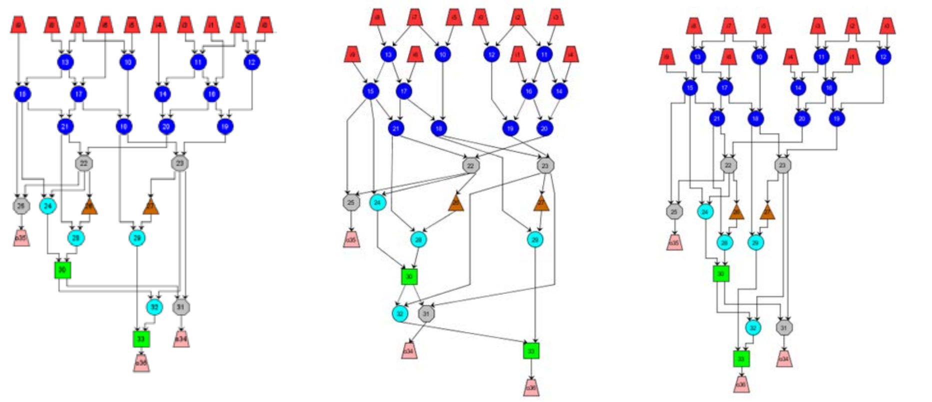
Orientation: TOP to BOTTOM
Layering: HIERARCHICAL TOPMOST
Style: MEDIAN SIMPLEX
Routing: ORTHOGONAL
Bend:
Min Node: Min Layer: Min Edge:
First Segment: Max Duration: Same Layer Edge

Core, [✓ Label], Sketch, [✓ Component], [✓ Self Loop], [✓ Orientation], [✓ Parallel Edge], Hide Node, [✓ Subgraph], False Crossings

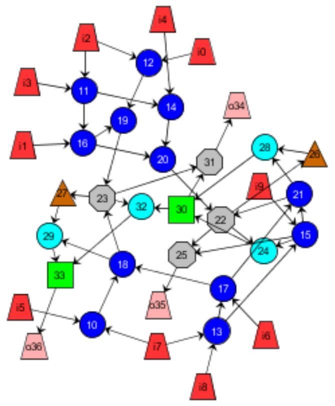
Common layout options

Use Case: Loading and Analyzing BENCH File

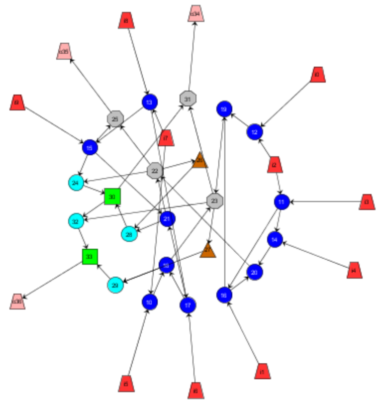
Basic Layout Types:



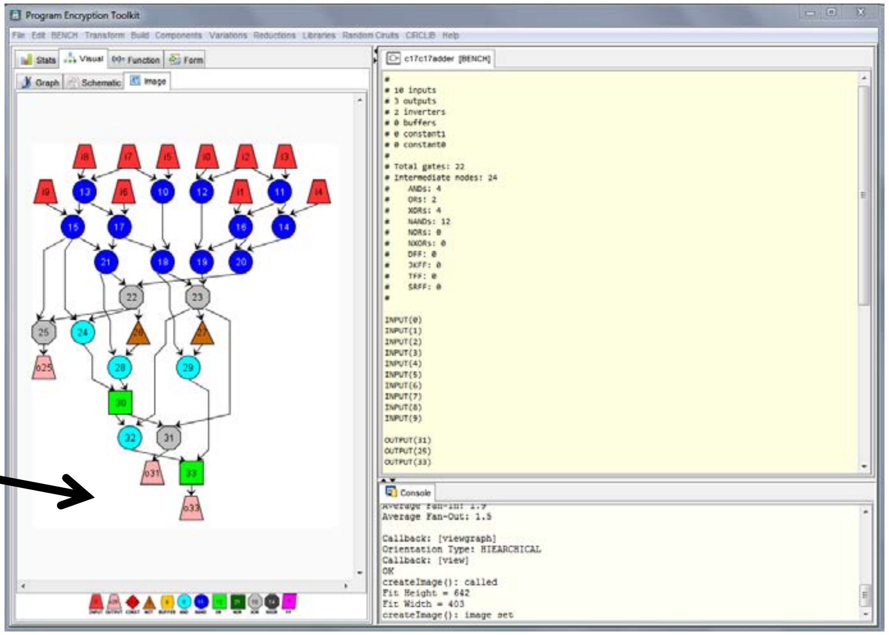
Hierarchical



Organic



Circular


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Circuits CIRCLIB Help
Stats Visual Function Form
Graph Schematic Image
c17c17adder [BENCH]
Console

```
#
# 10 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 24
#    ANDs: 4
#    ORs: 2
#    XORs: 4
#    NANDs: 12
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(26)
OUTPUT(33)
```

```
Average Fan-In: 1.9
Average Fan-Out: 1.5

Callback: [viewgraph]
Orientation Type: HIEARCHICAL
Callback: [view]
OK
createImage(): called
Fit Height = 642
Fit Width = 403
createImage(): image set
```

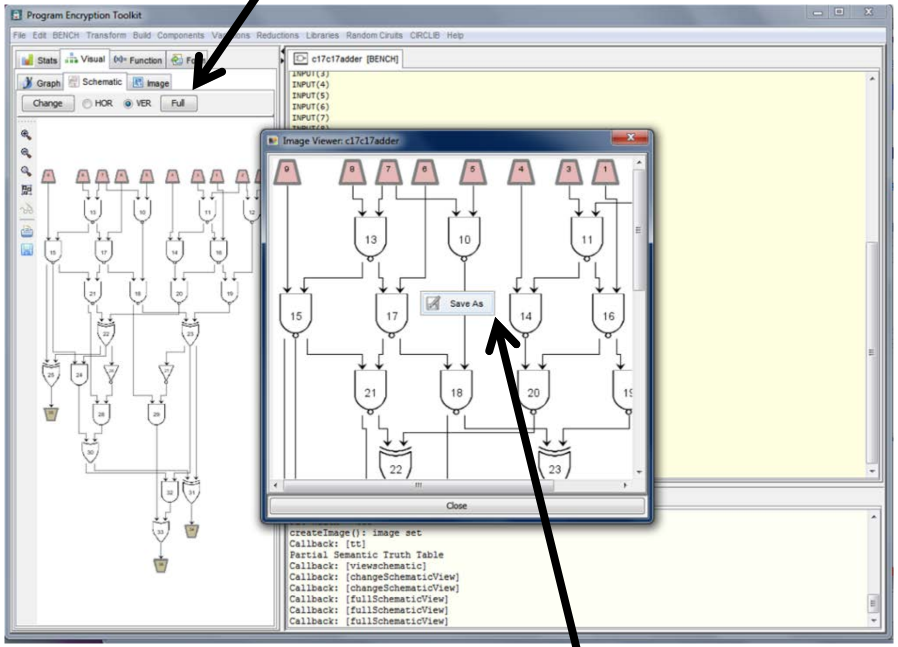
Right click support for image panels to save to file

5. BENCH->View Schematic

Mouse scroll wheel = zoom in/out

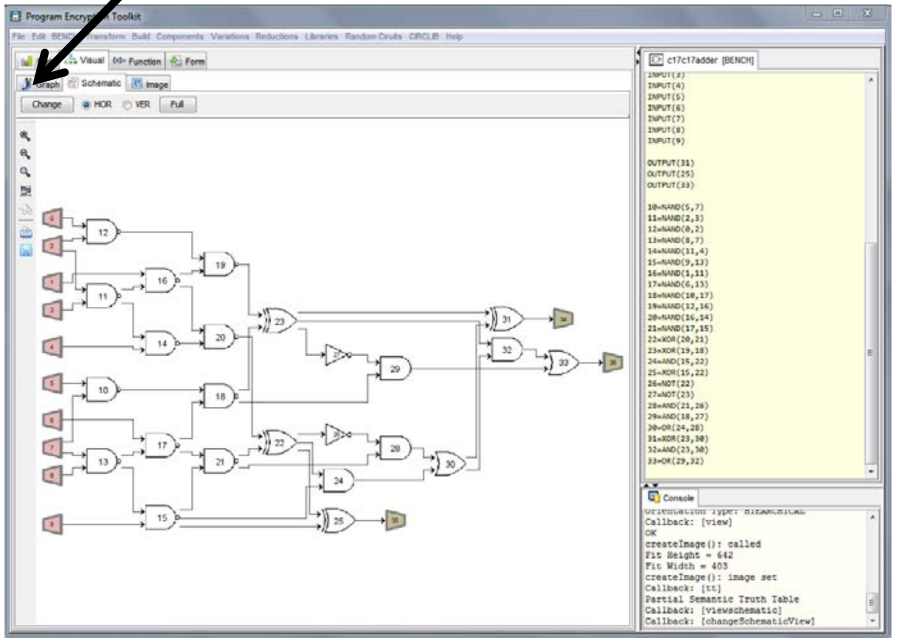
1. Click Full to bring up a full-size image

1. Choose orientation (HOR/VER)
2. Click Change


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function
Graph Schematic Image
Change HOR VER Full
c17c17adder [BENCH]
Image Viewer: c17c17adder
Save As
Close

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
```

```
createImage(): image set
Callback: [tt]
Partial Semantic Truth Table
Callback: [viewschematic]
Callback: [changeSchematicView]
Callback: [changeSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
```


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Visual Function Form
Graph Schematic Image
Change HOR VER Full
c17c17adder [BENCH]
Console

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Orientation Type: HIERARCHICAL
Callback: [view]
OK
createImage(): called
Fit Height = 642
Fit Width = 403
createImage(): image set
Callback: [tt]
Partial Semantic Truth Table
Callback: [viewschematic]
Callback: [changeSchematicView]
```

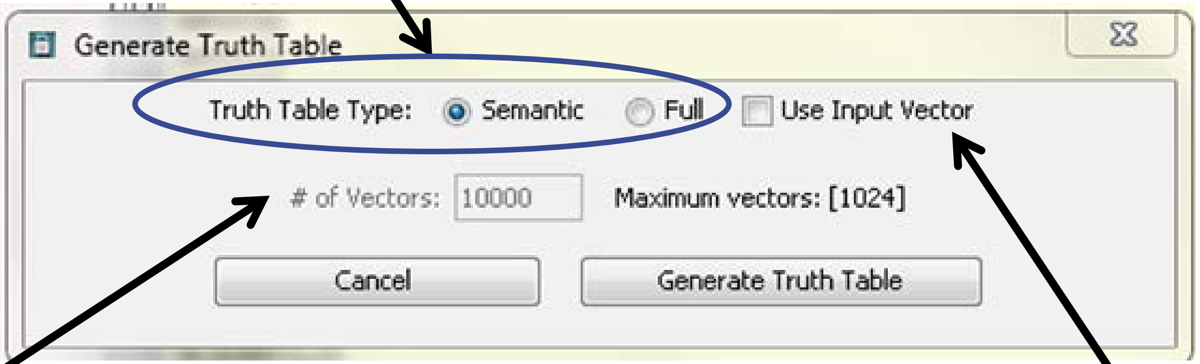
2. Right-click to save image from dialogue

6. BENCH->Generate Truth Table

NOTE: This generation is *O(2^n)* as it is based on all truth table rows, unless input vectors are used

Semantic Truth Table:
Show only inputs and output

Full Truth Table:
Show inputs, outputs, intermediate gate values


Generate Truth Table

Truth Table Type: [✓ Semantic] Full Use Input Vector

# of Vectors: 10000 Maximum vectors: [1024]

Number of input vectors:
<= 2^n, n = # inputs

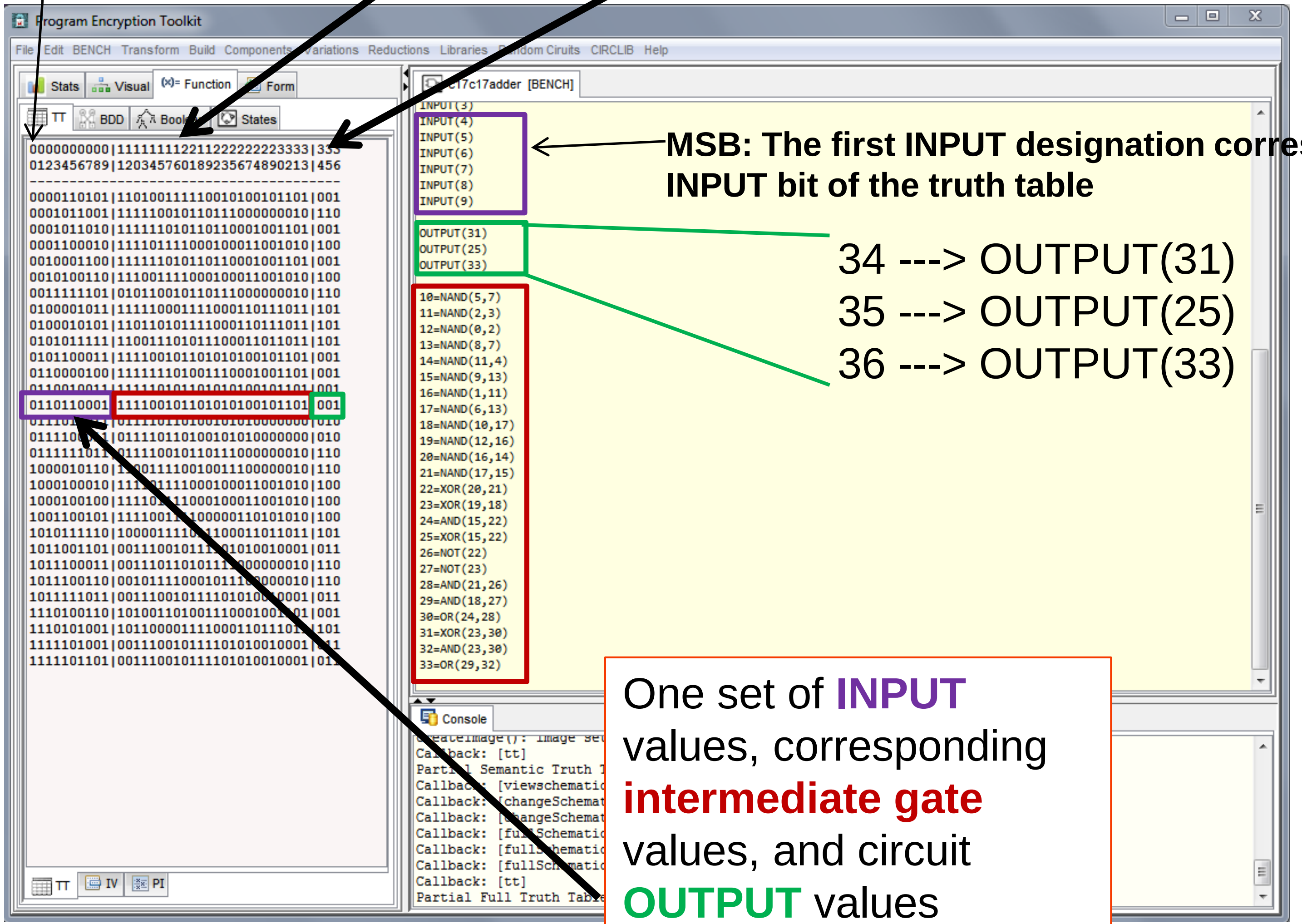
Input Vector: if selected, generates only a partial number of inputs and shows only those outputs (and intermediate results)

Truth Table: FULL, with input vector set

MSB

INTERMEDIATE gate signals

OUTPUT ports do not show up in BENCH text


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual (x)= Function Form
TT BDD Boolean States
TT IV PI
c17c17adder [BENCH]
Console

```
0000000000|1111111122112222222223333|333
0123456789|120345760189235674890213|456
-----------------------------------------
0000110101|1101001111100101001011101|001
0001011001|1111100101101110000000010|110
0001011010|1111110101101100010011101|001
0001100010|1111011110001000110010100|100
0010001100|1111110101101100010011101|001
0010100110|1110011110001000110010100|100
0011111101|0101100101101110000000010|110
0100001011|1111100011110001101110111|101
0100010101|1101101011110001101110111|101
0101011111|1100111010111000110110111|101
0101100011|1111001011010101001011101|001
0110000100|1111111010011100010011101|001
0110010011|1111101011010101001011101|001
0110110001|1111001011010101001011101|001
0111101[illegible]|0111101101001010100000000|010
0111100[illegible]1|0111101101001010100000000|010
0111111011|0111100101101110000000010|110
1000010110|1100111001001110000000010|110
1000100010|1111011110001000110010100|100
1000100100|1111011110001000110010100|100
1001100101|1111001111000001101010100|100
1010111110|1000011110011000110110111|101
1011001101|0011100101111010100100001|011
1011100011|0011101101011110000000010|110
1011100110|0010111100010111000000010|110
1011111011|0011100101111010100100001|011
1110100110|1010011010011100010011101|001
1110101001|1011000011110001101110111|101
1111101001|0011100101111010100100001|011
1111101101|0011100101111010100100001|011
```

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

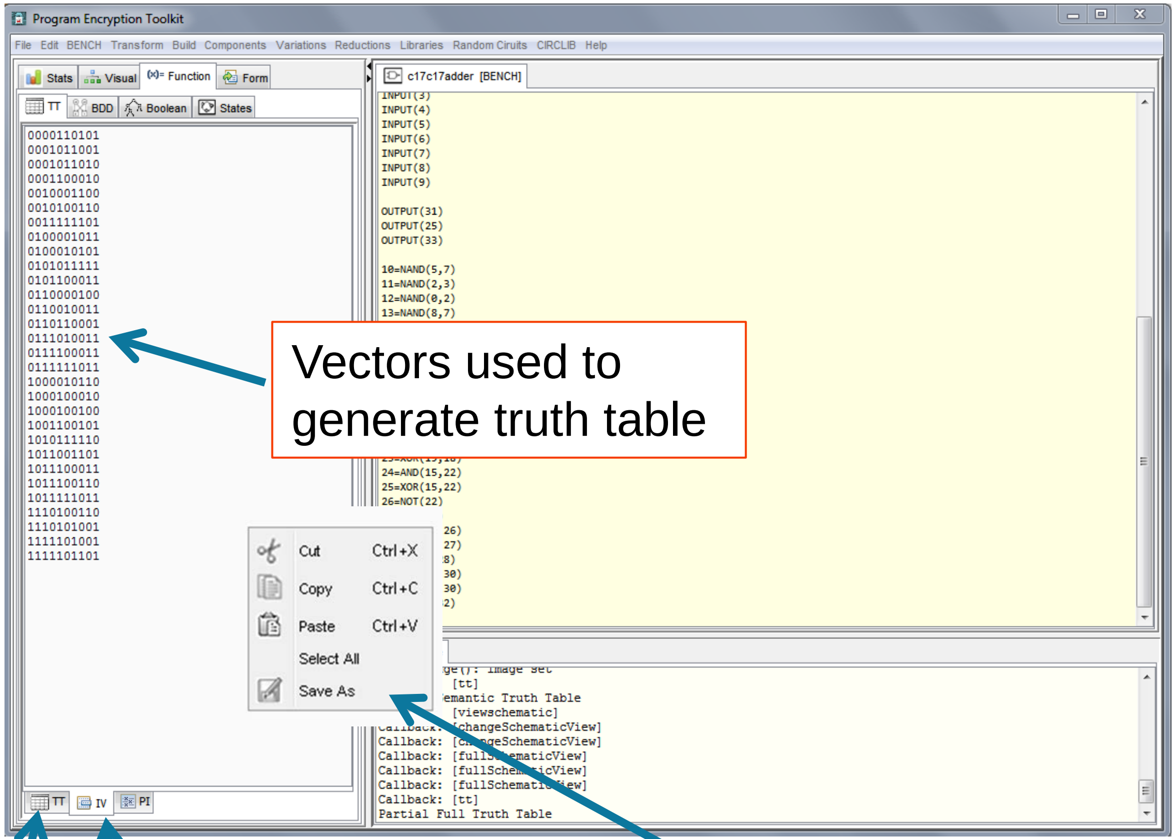
```
createImage(): Image set
Callback: [tt]
Partial Semantic Truth T
Callback: [viewschematic
Callback: [changeSchemat
Callback: [changeSchemat
Callback: [fullSchematic
Callback: [fullSchematic
Callback: [fullSchematic
Callback: [tt]
Partial Full Truth Table
```

MSB: The first INPUT designation corresponds to the first INPUT bit of the truth table

34 ---> OUTPUT(31)
35 ---> OUTPUT(25)
36 ---> OUTPUT(33)

One set of **INPUT** values, corresponding **intermediate gate** values, and circuit **OUTPUT** values

Input Vector


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats
Visual
(x)= Function
Form
TT
BDD
Boolean
States
c17c17adder [BENCH]
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All
Save As
TT
IV
PI

```
0000110101
0001011001
0001011010
0001100010
0010001100
0010100110
0011111101
0100001011
0100010101
0101011111
0101100011
0110000100
0110010011
0110110001
0111010011
0111100011
0111111011
1000010110
1000100010
1000100100
1001100101
1010111110
1011001101
1011100011
1011100110
1011111011
1110100110
1110101001
1111101001
1111101101
```

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)

24=AND(15,22)
25=XOR(15,22)
26=NOT(22)

26)
27)
8)
30)
30)
2)
```

```
ge(): image set
 [tt]
emantic Truth Table
 [viewschematic]
        [changeSchematicView]
Callback: [changeSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
```

Vectors used to generate truth table

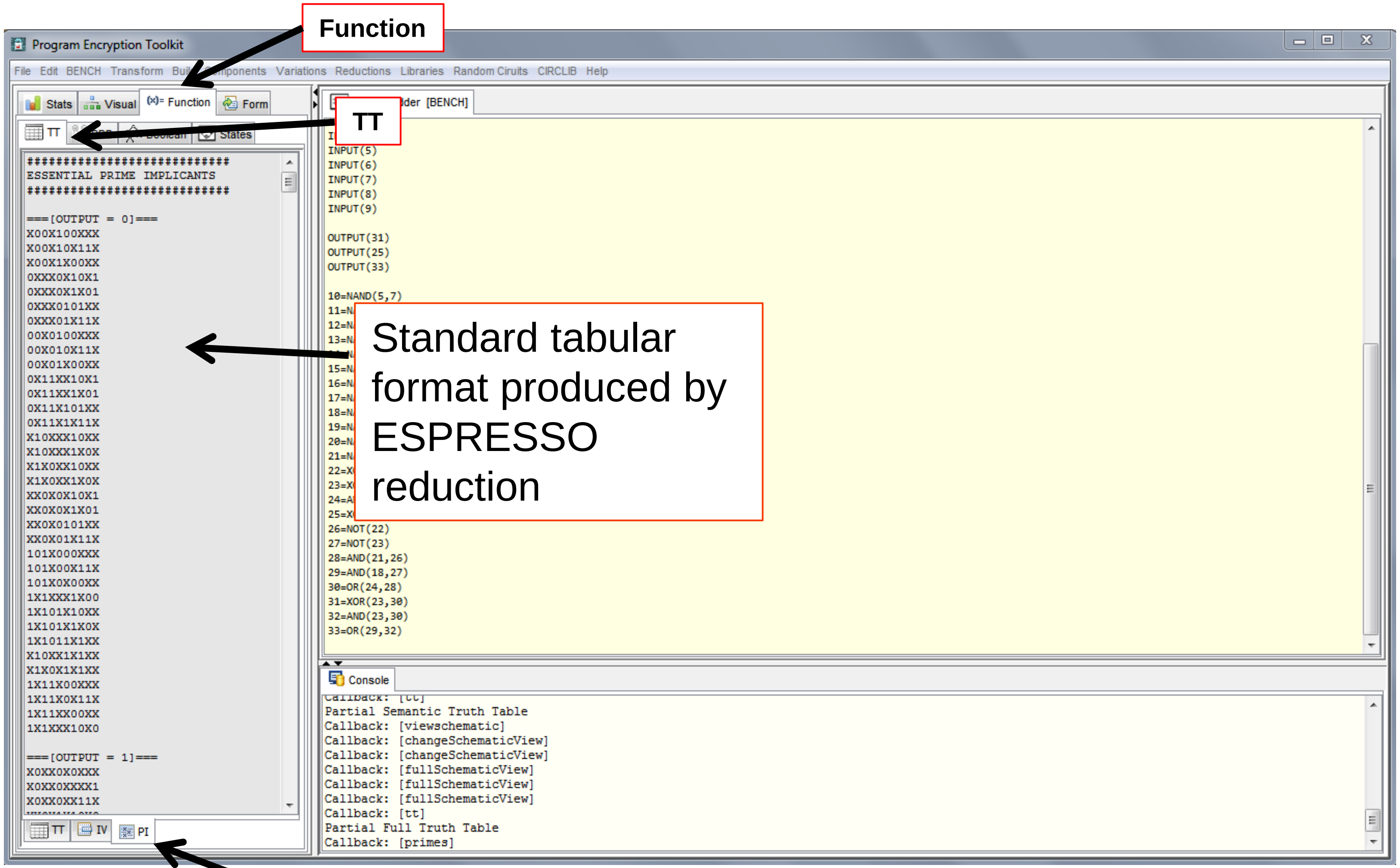
Prime Implicants

Input Vector tab

Truth Table tab

Right click for text panels: you can save any text to a file, do standard copy or cut/paste for editable text panels

7. BENCH->Implicants/Terms


Function
TT
Program Encryption Toolkit
File Edit BENCH Transform Bui… …mponents Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual (x)= Function Form
TT States
…der [BENCH]
Console
TT IV PI

```
############################
ESSENTIAL PRIME IMPLICANTS
############################

===[OUTPUT = 0]===
X00X100XXX
X00X10X11X
X00X1X00XX
0XXX0X10X1
0XXX0X1X01
0XXX0101XX
0XXX01X11X
00X0100XXX
00X010X11X
00X01X00XX
0X11XX10X1
0X11XX1X01
0X11X101XX
0X11X1X11X
X10XXX10XX
X10XXX1X0X
X1X0XX10XX
X1X0XX1X0X
XX0X0X10X1
XX0X0X1X01
XX0X0101XX
XX0X01X11X
101X000XXX
101X00X11X
101X0X00XX
1X1XXX1X00
1X101X10XX
1X101X1X0X
1X1011X1XX
X10XX1X1XX
X1X0X1X1XX
1X11X00XXX
1X11X0X11X
1X11XX00XX
1X1XXX10X0

===[OUTPUT = 1]===
X0XX0X0XXX
X0XX0XXXX1
X0XX0XX11X
```

```
I
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=N
12=N
13=N

15=N
16=N
17=N
18=N
19=N
20=N
21=N
22=X
23=X
24=A
25=X
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Callback: [tt]
Partial Semantic Truth Table
Callback: [viewschematic]
Callback: [changeSchematicView]
Callback: [changeSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
Callback: [primes]
```

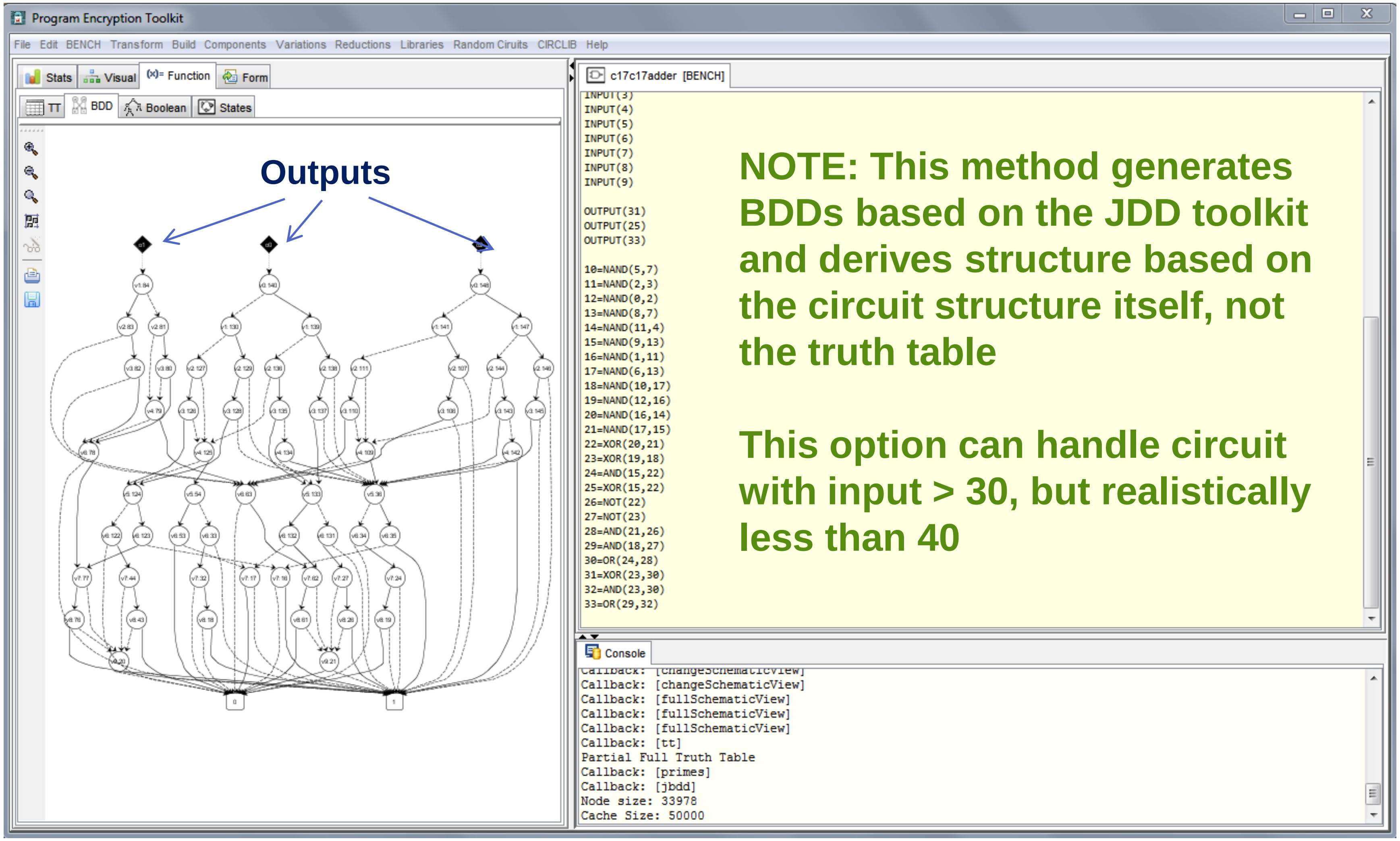
Standard tabular format produced by ESPRESSO reduction

Prime Implicants Tab

Use Case: Loading and Analyzing BENCH File

8. BENCH->MROBDD

Multi-Rooted Binary Decision Diagram


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
TT BDD Boolean States
Outputs
c17c17adder [BENCH]
Console

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

NOTE: This method generates BDDs based on the JDD toolkit and derives structure based on the circuit structure itself, not the truth table

This option can handle circuit with input > 30, but realistically less than 40

```
Callback: [changeSchematicView]
Callback: [changeSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
Callback: [primes]
Callback: [jbdd]
Node size: 33978
Cache Size: 50000
```

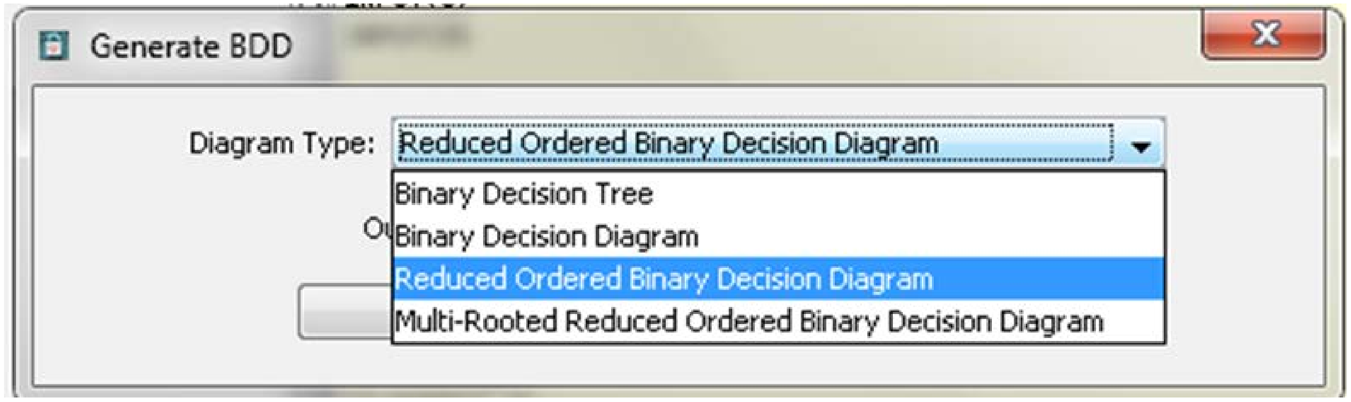
9. BENCH->Decision Trees

NOTE: This approach is *O(2^n)* as it is based on the full truth table, but is useful for illustrating BDD reduction on smaller functions

4 Main Types:

Per Single Output (must be chosen):
1-Binary Decision Tree
2-Binary Decision Diagram (BDD)
3-Reduced Ordered BDD (ROBDD)

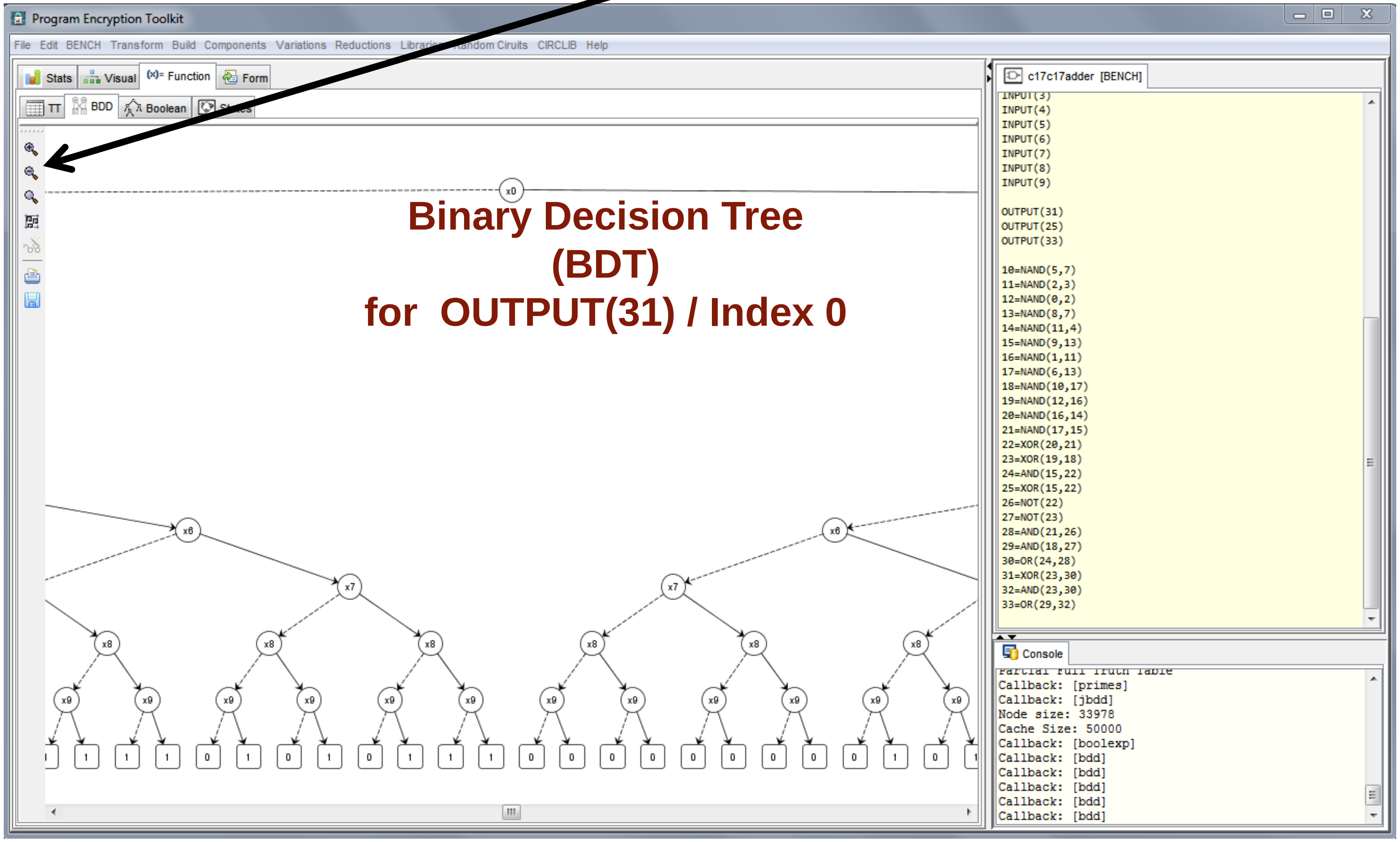
Shows all Outputs:
4-Multi-Rooted ROBDD (MROBDD)


Generate BDD
Diagram Type:
Reduced Ordered Binary Decision Diagram
Binary Decision Tree
Binary Decision Diagram
Reduced Ordered Binary Decision Diagram
Multi-Rooted Reduced Ordered Binary Decision Diagram

Decision Tree Types:

Note: for >5 inputs, these diagrams may be extremely large

Zoom, Magnifier, Save Image, and Print options

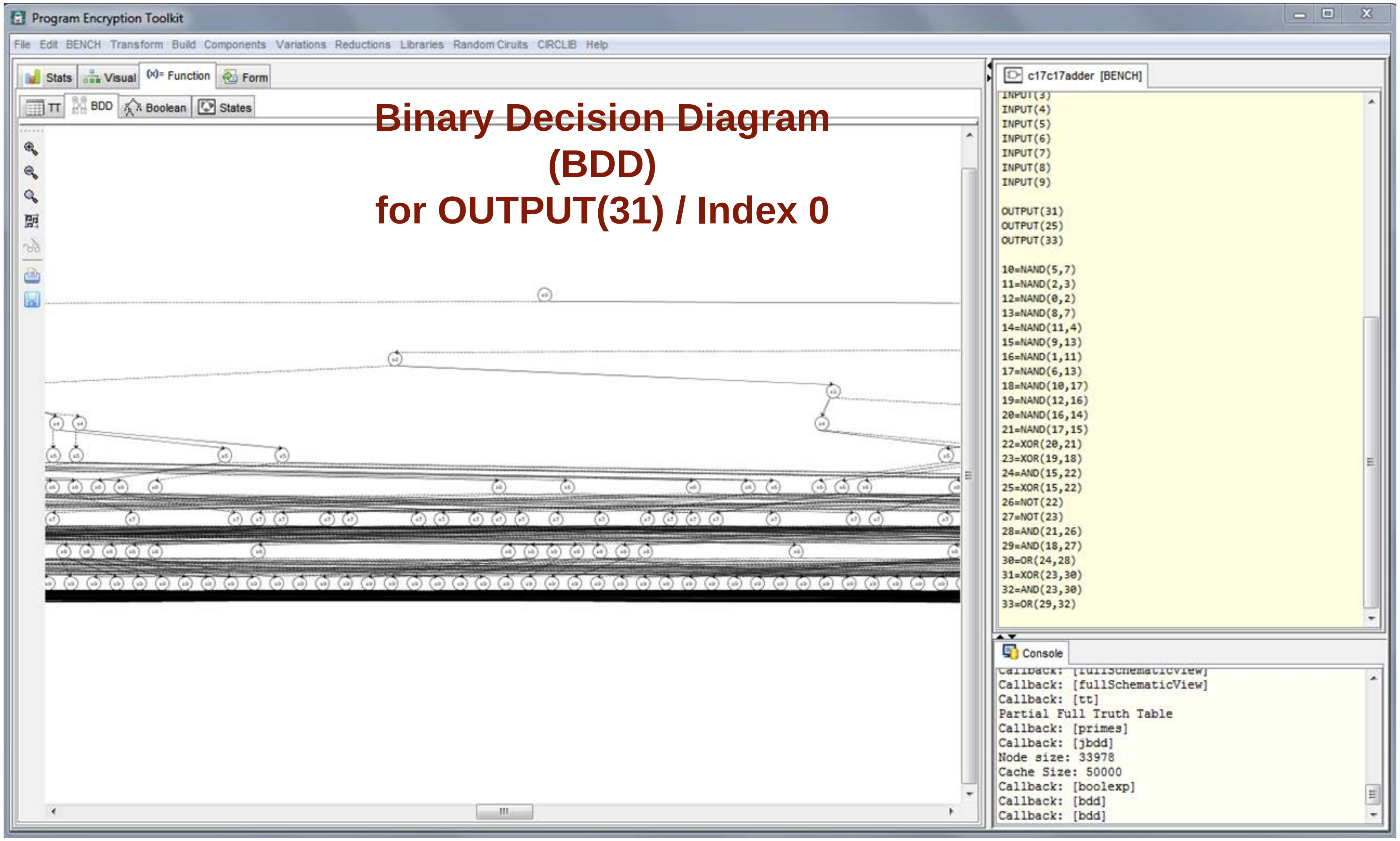

Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
TT BDD Boolean States
x0
x6
x7
x8
x9
0
1
c17c17adder [BENCH]
Console

Binary Decision Tree (BDT) for OUTPUT(31) / Index 0

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Partial Full Truth Table
Callback: [primes]
Callback: [jbdd]
Node size: 33978
Cache Size: 50000
Callback: [boolexp]
Callback: [bdd]
Callback: [bdd]
Callback: [bdd]
Callback: [bdd]
Callback: [bdd]
```

Decision Tree Types:


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
TT BDD Boolean States
c17c17adder [BENCH]
Console

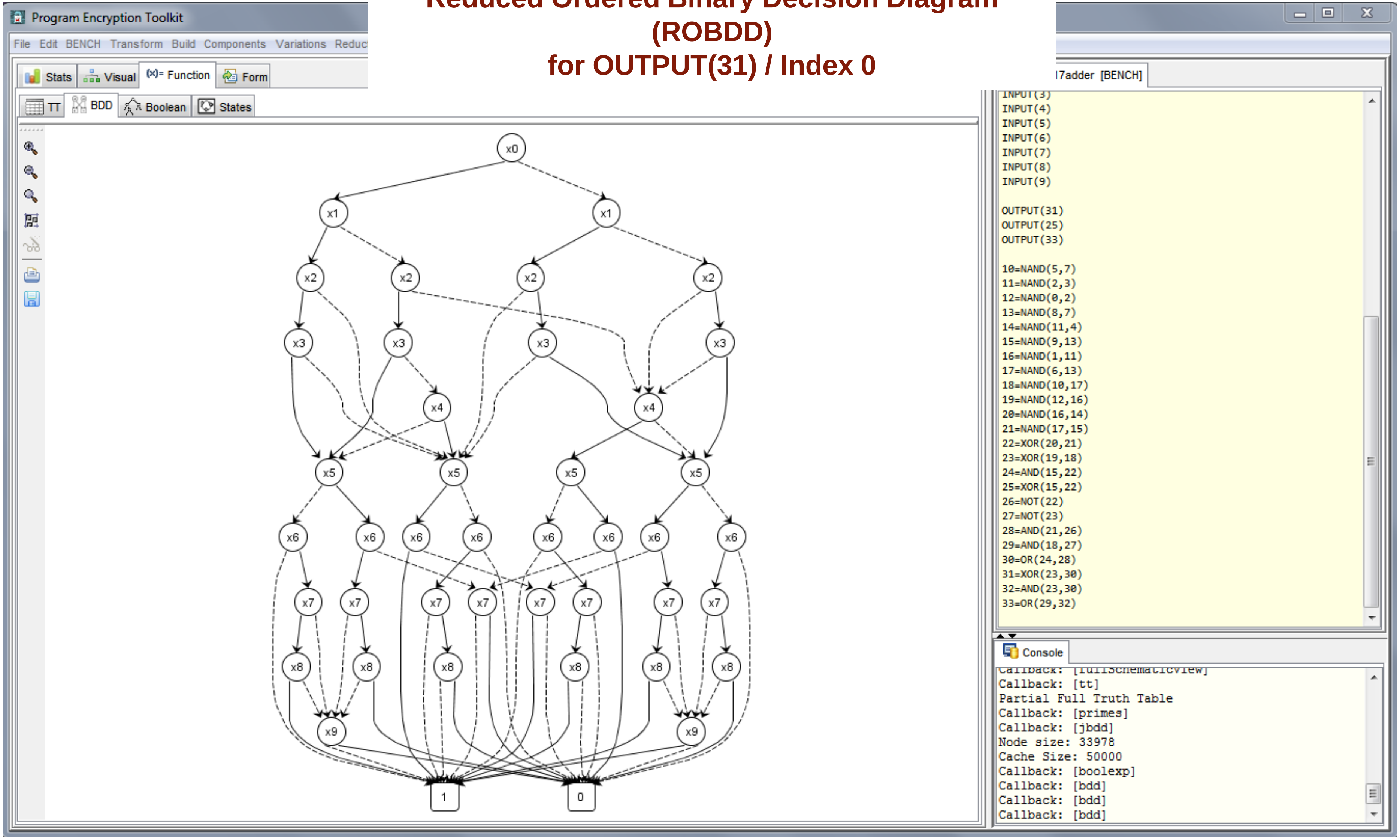
Binary Decision Diagram (BDD)
for OUTPUT(31) / Index 0

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
Callback: [primes]
Callback: [jbdd]
Node size: 33978
Cache Size: 50000
Callback: [boolexp]
Callback: [bdd]
Callback: [bdd]
```

Decision Tree Types:

Reduced Ordered Binary Decision Diagram (ROBDD)
for OUTPUT(31) / Index 0



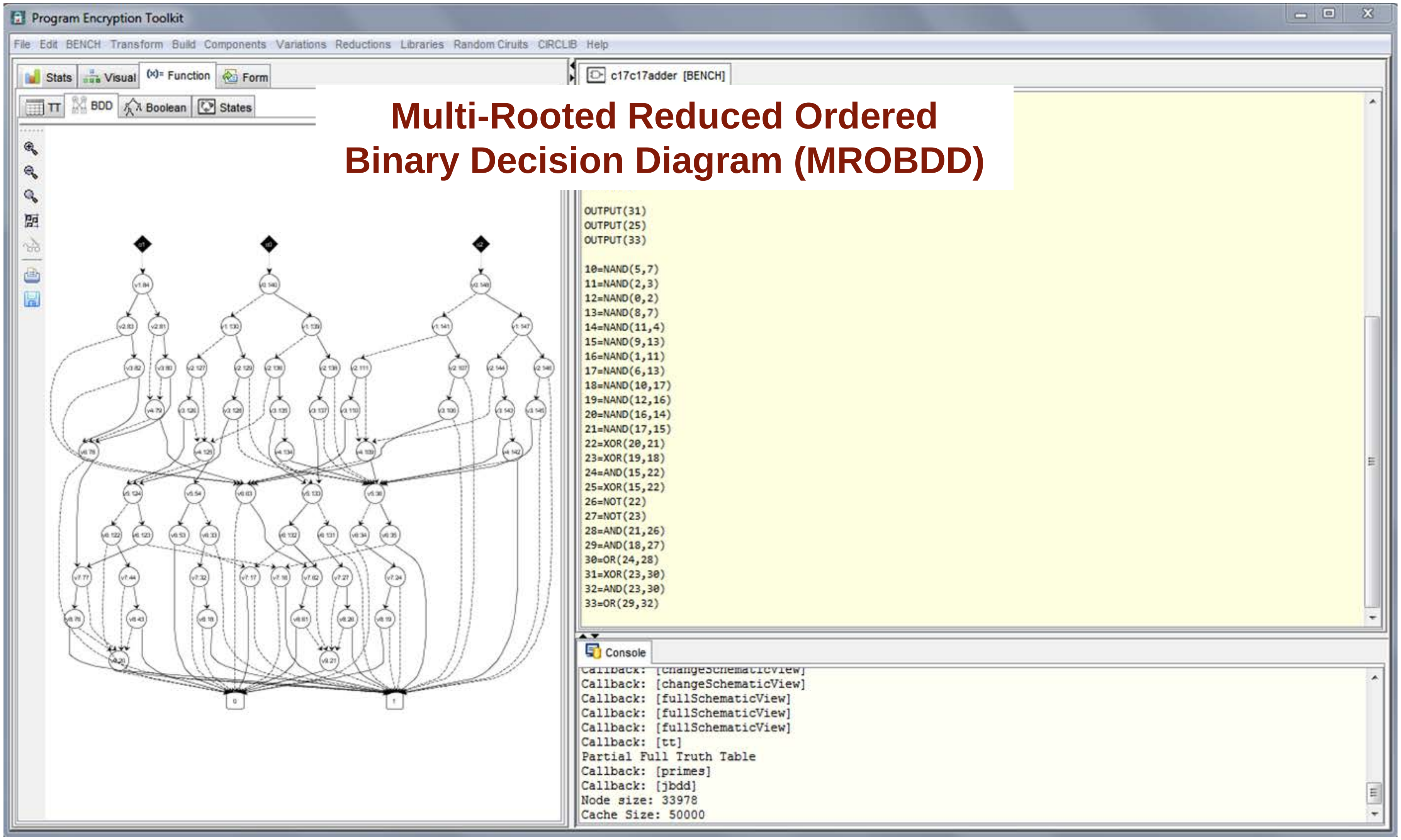
```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
Callback: [primes]
Callback: [jbdd]
Node size: 33978
Cache Size: 50000
Callback: [boolexp]
Callback: [bdd]
Callback: [bdd]
Callback: [bdd]
```

Use Case: Loading and Analyzing BENCH File

Decision Tree Types:

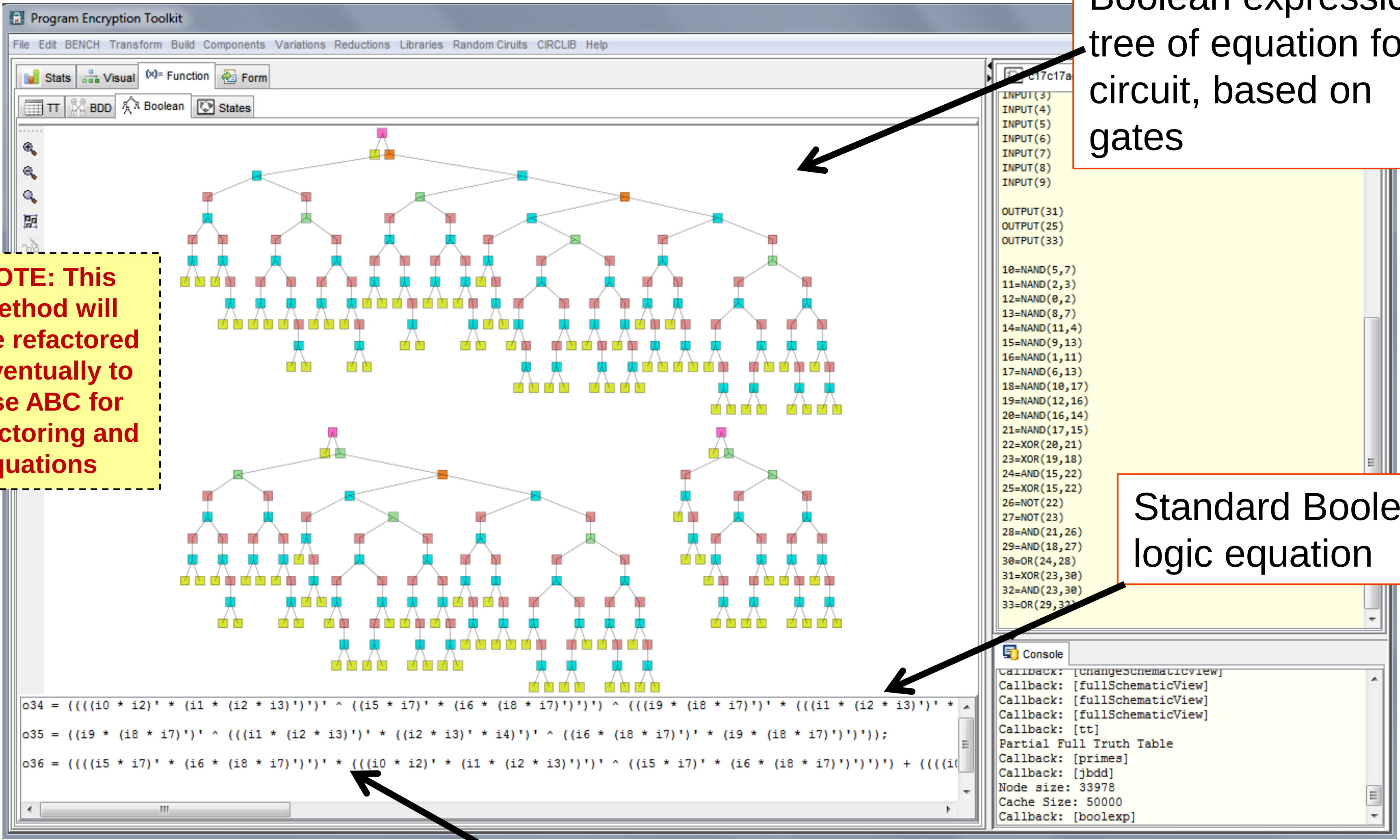
Multi-Rooted Reduced Ordered Binary Decision Diagram (MROBDD)


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
TT BDD Boolean States
c17c17adder [BENCH]
Console

```
OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Callback: [changeSchematicView]
Callback: [changeSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
Callback: [primes]
Callback: [jbdd]
Node size: 33978
Cache Size: 50000
```

10. BENCH->Boolean Expression Tree


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats
Visual
(x)= Function
Form
TT
BDD
Boolean
States
c17c17a

```
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

Console

```
Callback: [changeSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [fullSchematicView]
Callback: [tt]
Partial Full Truth Table
Callback: [primes]
Callback: [jbdd]
Node size: 33978
Cache Size: 50000
Callback: [boolexp]
```

```
o34 = ((((i0 * i2)' * (i1 * (i2 * i3)')')')' ^ ((i5 * i7)' * (i6 * (i8 * i7)')')') ^ (((i9 * (i8 * i7)')' * (((i1 * (i2 * i3)')' *

o35 = ((i9 * (i8 * i7)')' ^ (((i1 * (i2 * i3)')' * ((i2 * i3)' * i4)')' ^ ((i6 * (i8 * i7)')' * (i9 * (i8 * i7)')')')'));

o36 = ((((i5 * i7)' * (i6 * (i8 * i7)')')')' * (((i0 * i2)' * (i1 * (i2 * i3)')')' ^ ((i5 * i7)' * (i6 * (i8 * i7)')')')') + ((((i
```

Boolean expression tree of equation for circuit, based on gates

NOTE: This method will be refactored eventually to use ABC for factoring and equations

Standard Boolean logic equation

This string can be used in Logic Friday (copy/paste)

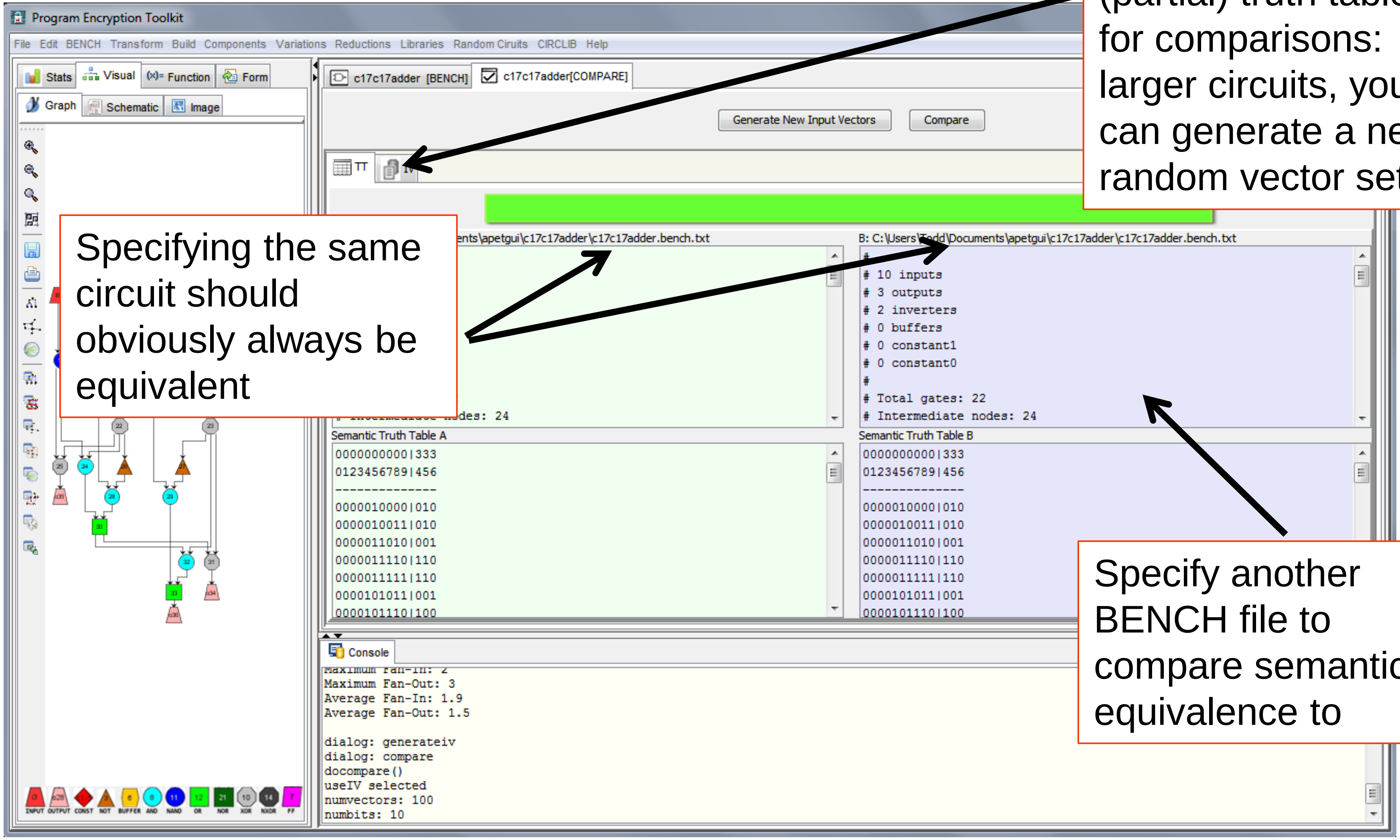
Use Case: Loading and Analyzing BENCH File

11. BENCH->Compare TT

Can use input vector (partial) truth table for comparisons: for larger circuits, you can generate a new random vector set

Specifying the same circuit should obviously always be equivalent

Specify another BENCH file to compare semantic equivalence to


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats
Visual
Function
Form
Graph
Schematic
Image
c17c17adder [BENCH]
c17c17adder[COMPARE]
Generate New Input Vectors
Compare
TT
IV
Console
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF

ents\apetgui\c17c17adder\c17c17adder.bench.txt

```
# Intermediate nodes: 24
```

B: C:\Users\Todd\Documents\apetgui\c17c17adder\c17c17adder.bench.txt

```
#
# 10 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 24
```

Semantic Truth Table A

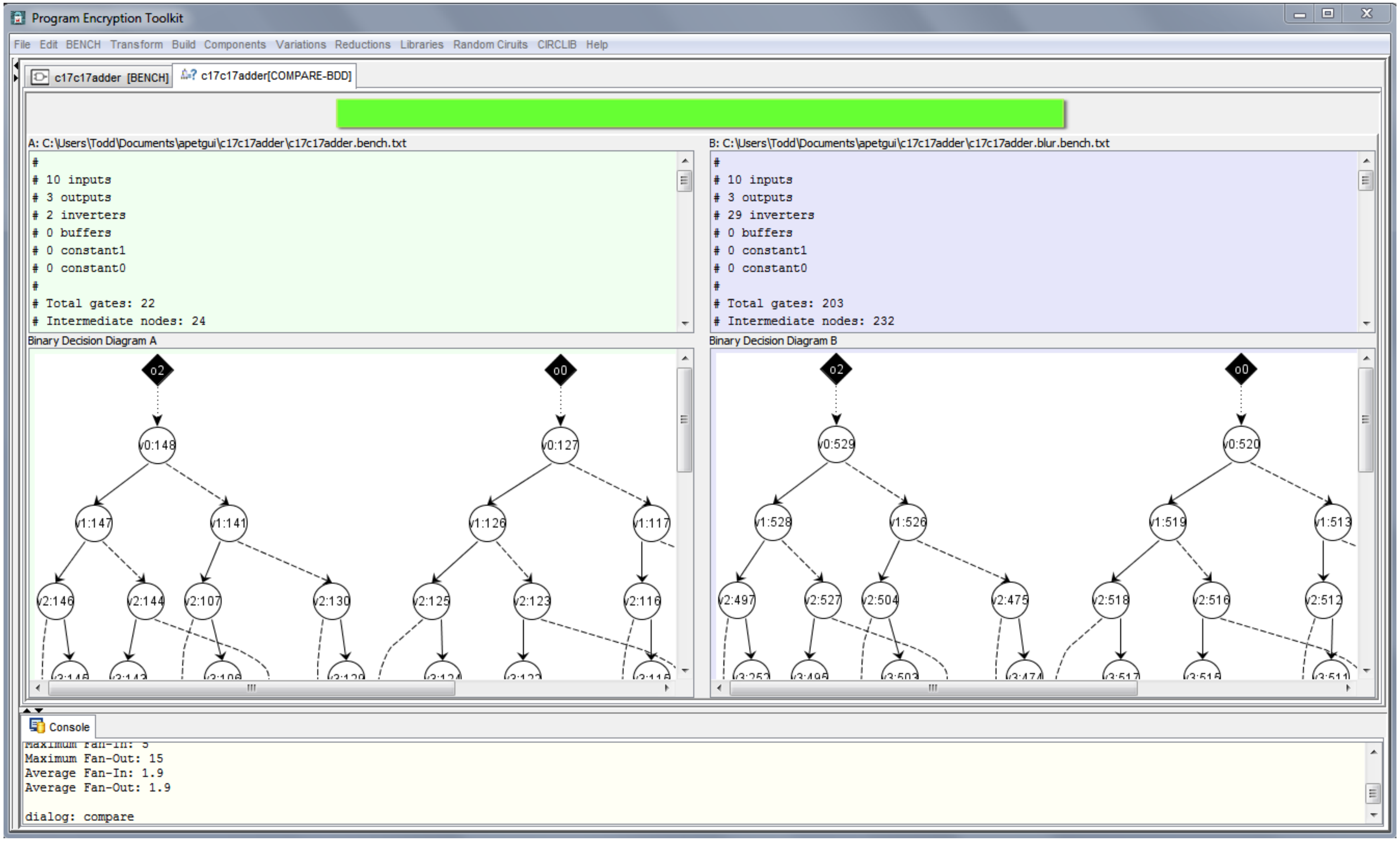
```
0000000000|333
0123456789|456
--------------
0000010000|010
0000010011|010
0000011010|001
0000011110|110
0000011111|110
0000101011|001
0000101110|100
```

Semantic Truth Table B

```
0000000000|333
0123456789|456
--------------
0000010000|010
0000010011|010
0000011010|001
0000011110|110
0000011111|110
0000101011|001
0000101110|100
```

```
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 1.9
Average Fan-Out: 1.5

dialog: generateiv
dialog: compare
docompare()
useIV selected
numvectors: 100
numbits: 10
```

12. BENCH->Compare BDD



A: C:\Users\Todd\Documents\apetgui\c17c17adder\c17c17adder.bench.txt

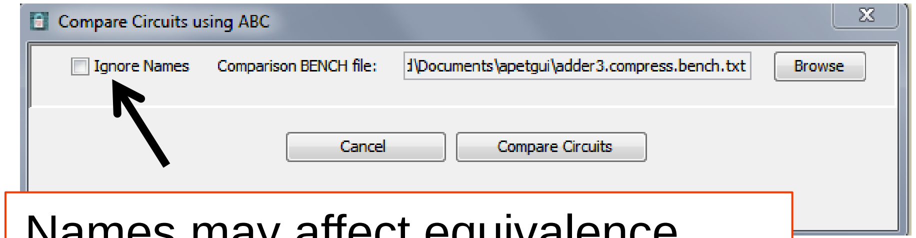
```
#
# 10 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 24
```

B: C:\Users\Todd\Documents\apetgui\c17c17adder\c17c17adder.blur.bench.txt

```
#
# 10 inputs
# 3 outputs
# 29 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 203
# Intermediate nodes: 232
```

```
Maximum Fan-Out: 15
Average Fan-In: 1.9
Average Fan-Out: 1.9

dialog: compare
```

13. BENCH->Compare ABC

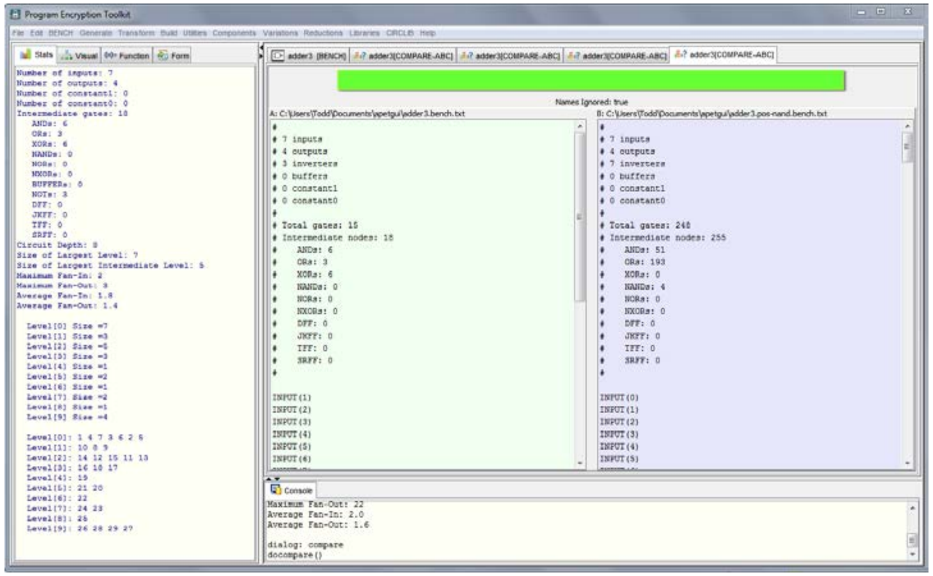

Compare Circuits using ABC
Ignore Names
Comparison BENCH file:
d\Documents\apetgui\adder3.compress.bench.txt
Browse
Cancel
Compare Circuits

Names may affect equivalence based on permuted input/output

Same comparison circuit

Ignore Names = true

Ignore Names = false


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats
Visual
Function
Form
adder3 [BENCH]
adder3[COMPARE-ABC]
adder3[COMPARE-ABC]
adder3[COMPARE-ABC]
adder3[COMPARE-ABC]
Console

```
Number of inputs: 7
Number of outputs: 4
Number of constant1: 0
Number of constant0: 0
Intermediate gates: 18
   ANDs: 6
   ORs: 3
   XORs: 6
   NANDs: 0
   NORs: 0
   NXORs: 0
   BUFFERs: 0
   NOTs: 3
   DFF: 0
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 8
Size of Largest Level: 7
Size of Largest Intermediate Level: 5
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 1.8
Average Fan-Out: 1.4

  Level[0] Size =7
  Level[1] Size =3
  Level[2] Size =5
  Level[3] Size =3
  Level[4] Size =1
  Level[5] Size =2
  Level[6] Size =1
  Level[7] Size =2
  Level[8] Size =1
  Level[9] Size =4

  Level[0]: 1 4 7 3 6 2 5
  Level[1]: 10 8 9
  Level[2]: 14 12 15 11 13
  Level[3]: 16 18 17
  Level[4]: 19
  Level[5]: 21 20
  Level[6]: 22
  Level[7]: 24 23
  Level[8]: 25
  Level[9]: 26 28 29 27
```

Names Ignored: true

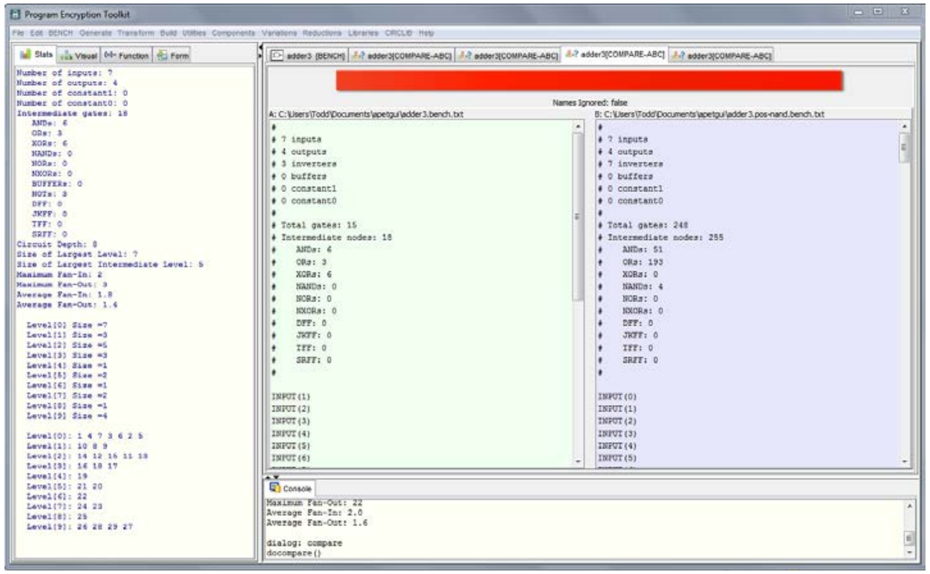
A: C:\Users\Todd\Documents\apetgui\adder3.bench.txt

```
#
# 7 inputs
# 4 outputs
# 3 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 15
# Intermediate nodes: 18
#    ANDs: 6
#    ORs: 3
#    XORs: 6
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
```

B: C:\Users\Todd\Documents\apetgui\adder3.pos-nand.bench.txt

```
#
# 7 inputs
# 4 outputs
# 7 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 248
# Intermediate nodes: 255
#    ANDs: 51
#    ORs: 193
#    XORs: 0
#    NANDs: 4
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
```

```
Maximum Fan-Out: 22
Average Fan-In: 2.0
Average Fan-Out: 1.6

dialog: compare
docompare()
```


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats
Visual
Function
Form
adder3 [BENCH]
adder3[COMPARE-ABC]
adder3[COMPARE-ABC]
adder3[COMPARE-ABC]
adder3[COMPARE-ABC]
Console

```
Number of inputs: 7
Number of outputs: 4
Number of constant1: 0
Number of constant0: 0
Intermediate gates: 18
   ANDs: 6
   ORs: 3
   XORs: 6
   NANDs: 0
   NORs: 0
   NXORs: 0
   BUFFERs: 0
   NOTs: 3
   DFF: 0
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 8
Size of Largest Level: 7
Size of Largest Intermediate Level: 5
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 1.8
Average Fan-Out: 1.4

  Level[0] Size =7
  Level[1] Size =3
  Level[2] Size =5
  Level[3] Size =3
  Level[4] Size =1
  Level[5] Size =2
  Level[6] Size =1
  Level[7] Size =2
  Level[8] Size =1
  Level[9] Size =4

  Level[0]: 1 4 7 3 6 2 5
  Level[1]: 10 8 9
  Level[2]: 14 12 15 11 13
  Level[3]: 16 18 17
  Level[4]: 19
  Level[5]: 21 20
  Level[6]: 22
  Level[7]: 24 23
  Level[8]: 25
  Level[9]: 26 28 29 27
```

Names Ignored: false

A: C:\Users\Todd\Documents\apetgui\adder3.bench.txt

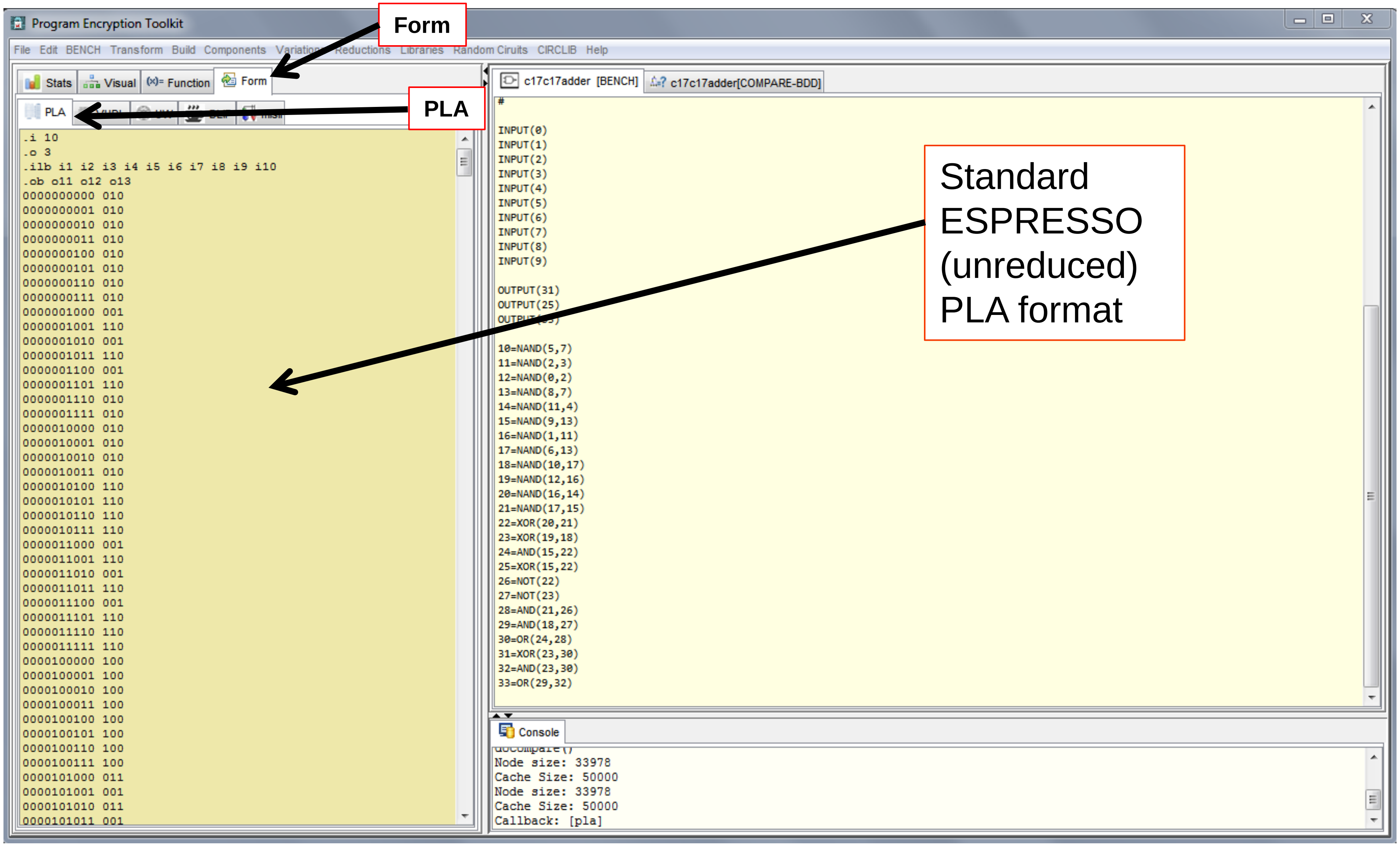
```
#
# 7 inputs
# 4 outputs
# 3 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 15
# Intermediate nodes: 18
#    ANDs: 6
#    ORs: 3
#    XORs: 6
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
```

B: C:\Users\Todd\Documents\apetgui\adder3.pos-nand.bench.txt

```
#
# 7 inputs
# 4 outputs
# 7 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 248
# Intermediate nodes: 255
#    ANDs: 51
#    ORs: 193
#    XORs: 0
#    NANDs: 4
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
```

```
Maximum Fan-Out: 22
Average Fan-In: 2.0
Average Fan-Out: 1.6

dialog: compare
docompare()
```

14. Generate ->Generate PLA

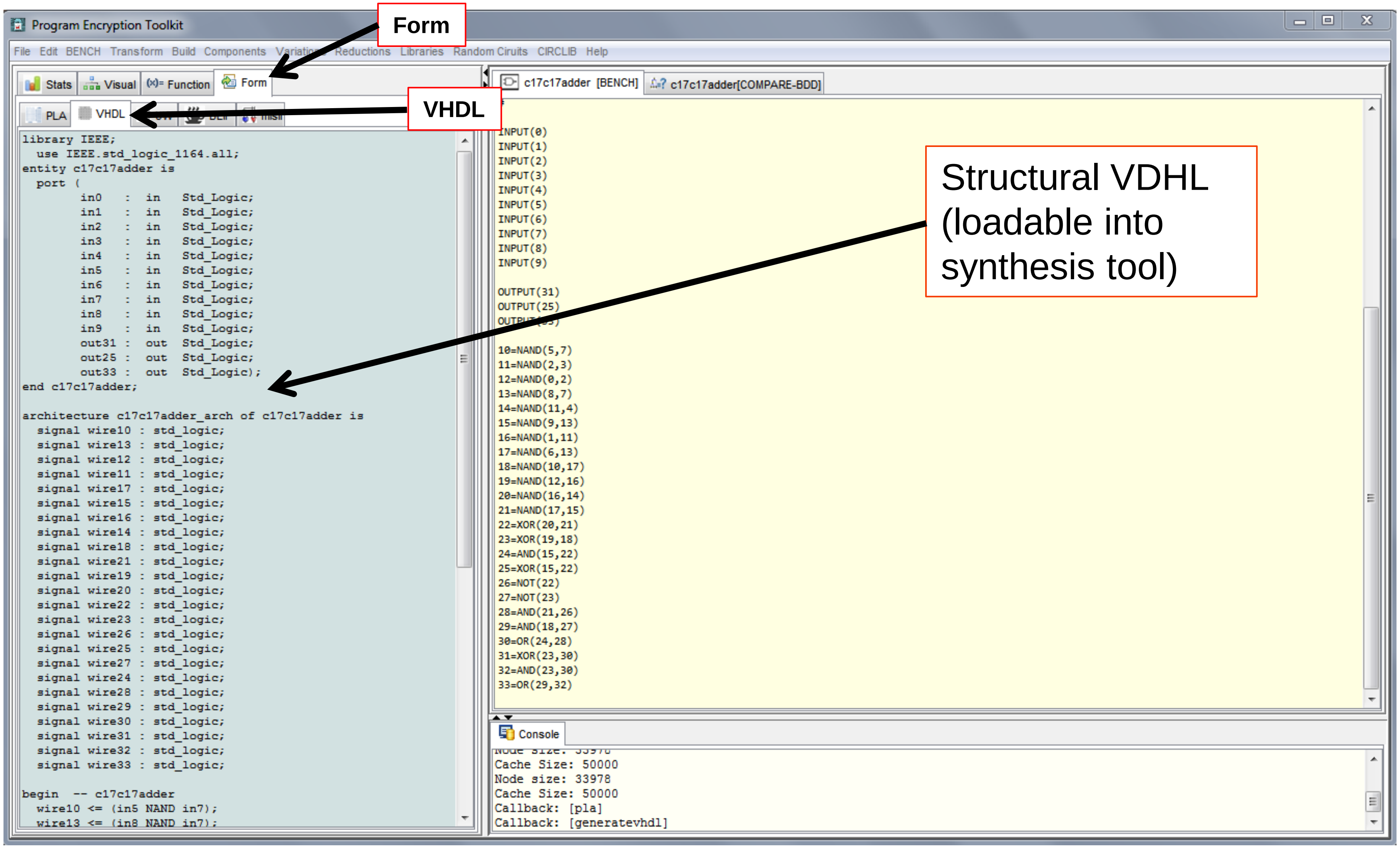

Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
PLA
c17c17adder [BENCH]
c17c17adder[COMPARE-BDD]
Console
Form
PLA
Standard ESPRESSO (unreduced) PLA format

```
.i 10
.o 3
.ilb i1 i2 i3 i4 i5 i6 i7 i8 i9 i10
.ob o11 o12 o13
0000000000 010
0000000001 010
0000000010 010
0000000011 010
0000000100 010
0000000101 010
0000000110 010
0000000111 010
0000001000 001
0000001001 110
0000001010 001
0000001011 110
0000001100 001
0000001101 110
0000001110 010
0000001111 010
0000010000 010
0000010001 010
0000010010 010
0000010011 010
0000010100 110
0000010101 110
0000010110 110
0000010111 110
0000011000 001
0000011001 110
0000011010 001
0000011011 110
0000011100 001
0000011101 110
0000011110 110
0000011111 110
0000100000 100
0000100001 100
0000100010 100
0000100011 100
0000100100 100
0000100101 100
0000100110 100
0000100111 100
0000101000 011
0000101001 001
0000101010 011
0000101011 001
```

```
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
ddcompare()
Node size: 33978
Cache Size: 50000
Node size: 33978
Cache Size: 50000
Callback: [pla]
```

15. Generate ->Generate VHDL


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
PLA VHDL
Form
VHDL
c17c17adder [BENCH]
c17c17adder[COMPARE-BDD]
Console

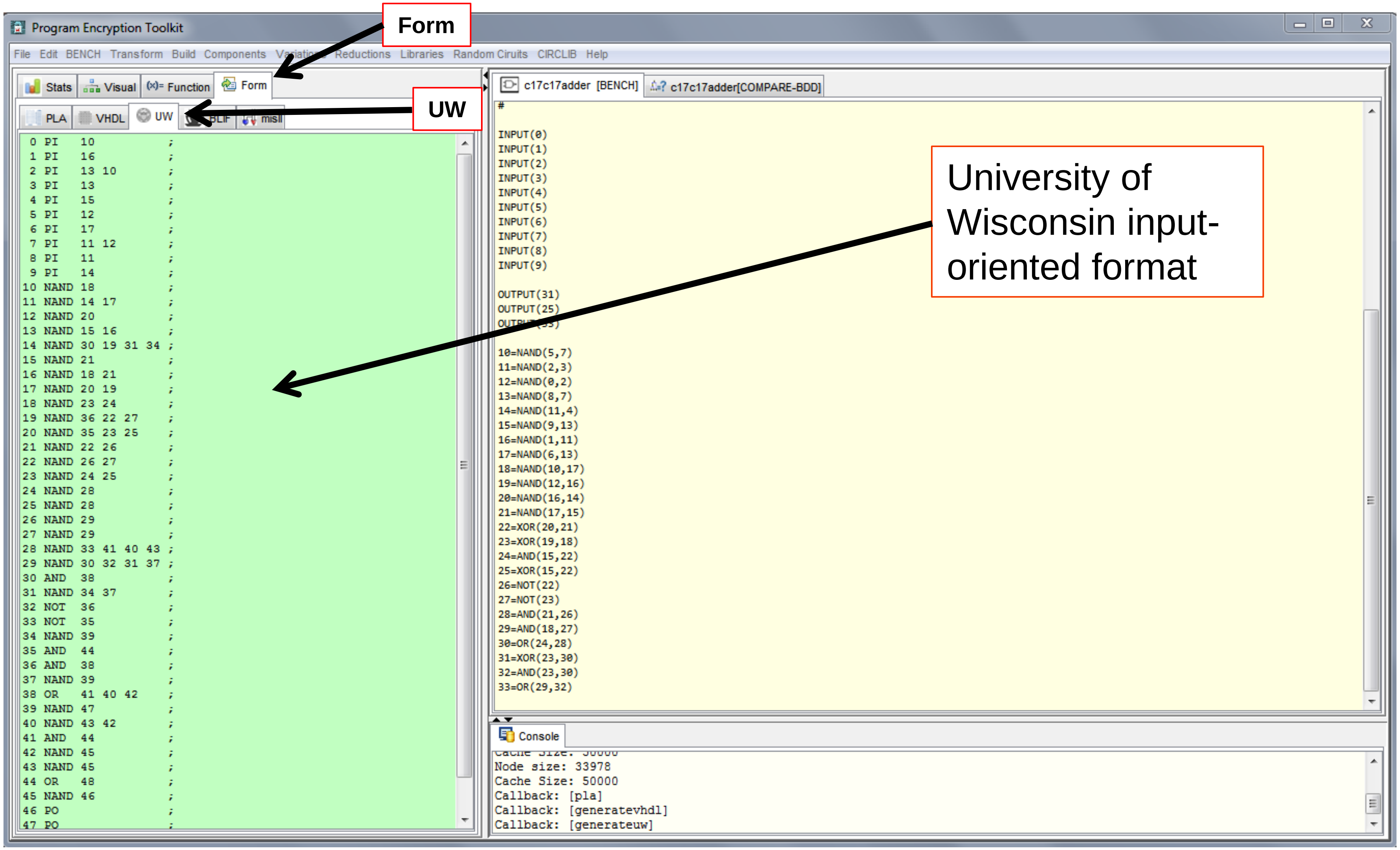
Structural VDHL (loadable into synthesis tool)

```
library IEEE;
  use IEEE.std_logic_1164.all;
entity c17c17adder is
  port (
        in0    :  in    Std_Logic;
        in1    :  in    Std_Logic;
        in2    :  in    Std_Logic;
        in3    :  in    Std_Logic;
        in4    :  in    Std_Logic;
        in5    :  in    Std_Logic;
        in6    :  in    Std_Logic;
        in7    :  in    Std_Logic;
        in8    :  in    Std_Logic;
        in9    :  in    Std_Logic;
        out31  :  out   Std_Logic;
        out25  :  out   Std_Logic;
        out33  :  out   Std_Logic);
end c17c17adder;

architecture c17c17adder_arch of c17c17adder is
  signal wire10 : std_logic;
  signal wire13 : std_logic;
  signal wire12 : std_logic;
  signal wire11 : std_logic;
  signal wire17 : std_logic;
  signal wire15 : std_logic;
  signal wire16 : std_logic;
  signal wire14 : std_logic;
  signal wire18 : std_logic;
  signal wire21 : std_logic;
  signal wire19 : std_logic;
  signal wire20 : std_logic;
  signal wire22 : std_logic;
  signal wire23 : std_logic;
  signal wire26 : std_logic;
  signal wire25 : std_logic;
  signal wire27 : std_logic;
  signal wire24 : std_logic;
  signal wire28 : std_logic;
  signal wire29 : std_logic;
  signal wire30 : std_logic;
  signal wire31 : std_logic;
  signal wire32 : std_logic;
  signal wire33 : std_logic;

begin  -- c17c17adder
  wire10 <= (in5 NAND in7);
  wire13 <= (in8 NAND in7);
```

```
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Node size: 33978
Cache Size: 50000
Node size: 33978
Cache Size: 50000
Callback: [pla]
Callback: [generatevhdl]
```

16. Generate ->Generate UW

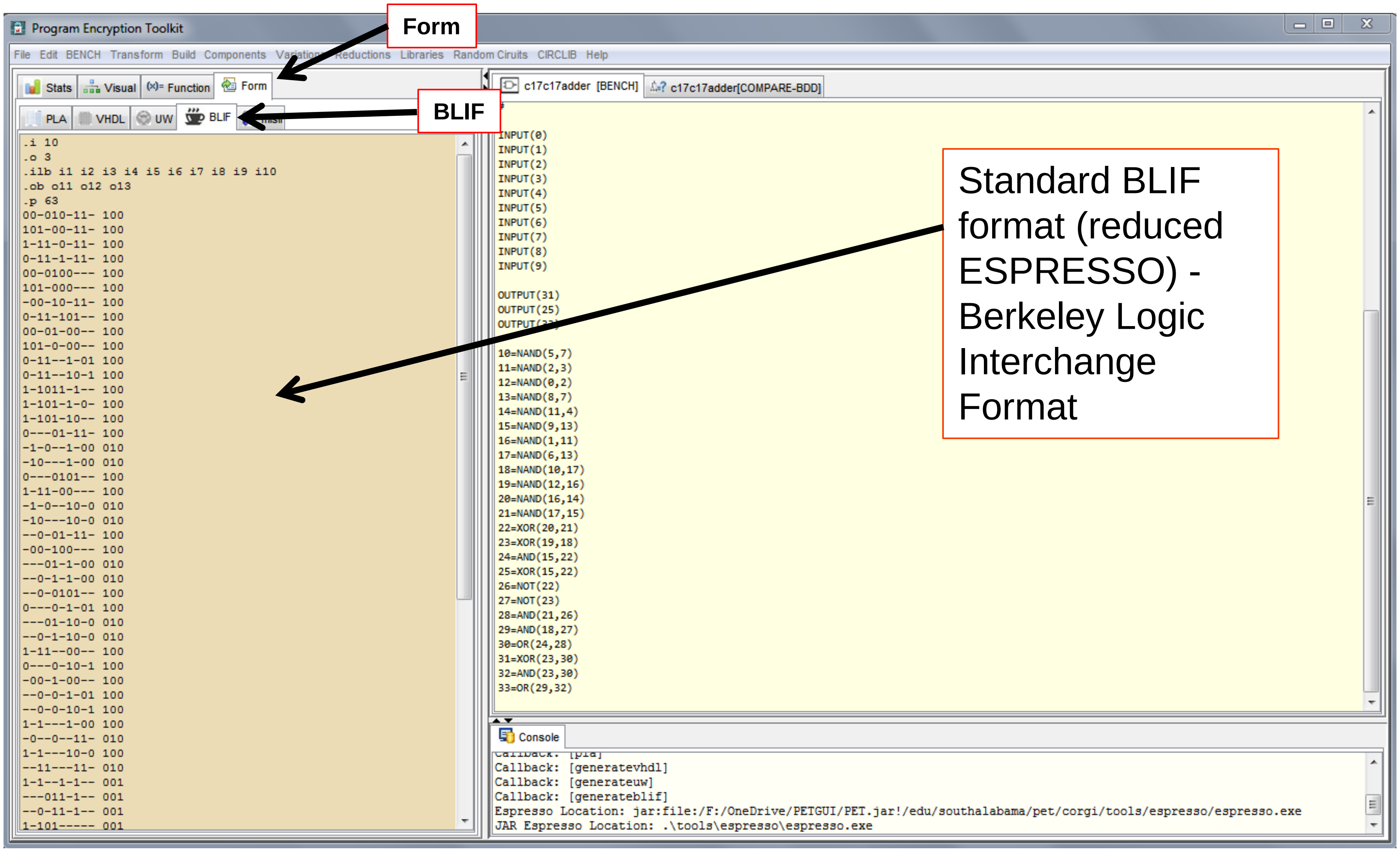

Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
PLA VHDL UW BLIF misII
c17c17adder [BENCH]
c17c17adder[COMPARE-BDD]
Console
Form
UW
University of Wisconsin input-oriented format

```
 0 PI   10          ;
 1 PI   16          ;
 2 PI   13 10       ;
 3 PI   13          ;
 4 PI   15          ;
 5 PI   12          ;
 6 PI   17          ;
 7 PI   11 12       ;
 8 PI   11          ;
 9 PI   14          ;
10 NAND 18          ;
11 NAND 14 17       ;
12 NAND 20          ;
13 NAND 15 16       ;
14 NAND 30 19 31 34 ;
15 NAND 21          ;
16 NAND 18 21       ;
17 NAND 20 19       ;
18 NAND 23 24       ;
19 NAND 36 22 27    ;
20 NAND 35 23 25    ;
21 NAND 22 26       ;
22 NAND 26 27       ;
23 NAND 24 25       ;
24 NAND 28          ;
25 NAND 28          ;
26 NAND 29          ;
27 NAND 29          ;
28 NAND 33 41 40 43 ;
29 NAND 30 32 31 37 ;
30 AND  38          ;
31 NAND 34 37       ;
32 NOT  36          ;
33 NOT  35          ;
34 NAND 39          ;
35 AND  44          ;
36 AND  38          ;
37 NAND 39          ;
38 OR   41 40 42    ;
39 NAND 47          ;
40 NAND 43 42       ;
41 AND  44          ;
42 NAND 45          ;
43 NAND 45          ;
44 OR   48          ;
45 NAND 46          ;
46 PO               ;
47 PO               ;
```

```
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT(33)

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Cache Size: 50000
Node size: 33978
Cache Size: 50000
Callback: [pla]
Callback: [generatevhdl]
Callback: [generateuw]
```

17. Generate ->Generate BLIF (Espresso)


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual (x)= Function Form
PLA VHDL UW BLIF
c17c17adder [BENCH] c17c17adder[COMPARE-BDD]
Console
Form
BLIF

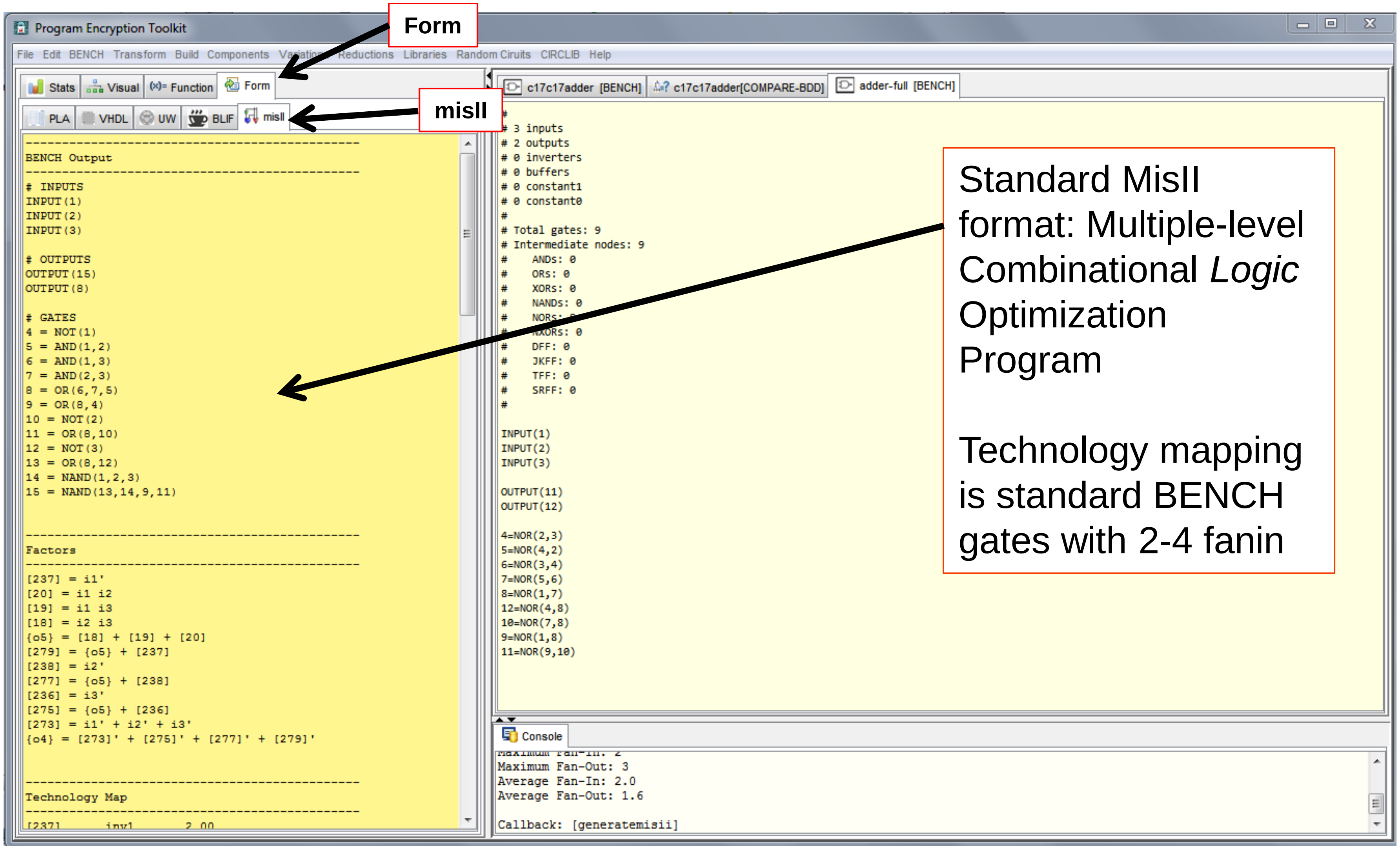
```
.i 10
.o 3
.ilb i1 i2 i3 i4 i5 i6 i7 i8 i9 i10
.ob o11 o12 o13
.p 63
00-010-11- 100
101-00-11- 100
1-11-0-11- 100
0-11-1-11- 100
00-0100--- 100
101-000--- 100
-00-10-11- 100
0-11-101-- 100
00-01-00-- 100
101-0-00-- 100
0-11--1-01 100
0-11--10-1 100
1-1011-1-- 100
1-101-1-0- 100
1-101-10-- 100
0---01-11- 100
-1-0--1-00 010
-10---1-00 010
0---0101-- 100
1-11-00--- 100
-1-0--10-0 010
-10---10-0 010
--0-01-11- 100
-00-100--- 100
---01-1-00 010
--0-1-1-00 010
--0-0101-- 100
0---0-1-01 100
---01-10-0 010
--0-1-10-0 010
1-11--00-- 100
0---0-10-1 100
-00-1-00-- 100
--0-0-1-01 100
--0-0-10-1 100
1-1---1-00 100
-0--0--11- 010
1-1---10-0 100
--11---11- 010
1-1--1-1-- 001
---011-1-- 001
--0-11-1-- 001
1-101----- 001
```

```
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(31)
OUTPUT(25)
OUTPUT([illegible]

10=NAND(5,7)
11=NAND(2,3)
12=NAND(0,2)
13=NAND(8,7)
14=NAND(11,4)
15=NAND(9,13)
16=NAND(1,11)
17=NAND(6,13)
18=NAND(10,17)
19=NAND(12,16)
20=NAND(16,14)
21=NAND(17,15)
22=XOR(20,21)
23=XOR(19,18)
24=AND(15,22)
25=XOR(15,22)
26=NOT(22)
27=NOT(23)
28=AND(21,26)
29=AND(18,27)
30=OR(24,28)
31=XOR(23,30)
32=AND(23,30)
33=OR(29,32)
```

```
Callback: [pla]
Callback: [generatevhdl]
Callback: [generateuw]
Callback: [generateblif]
Espresso Location: jar:file:/F:/OneDrive/PETGUI/PET.jar!/edu/southalabama/pet/corgi/tools/espresso/espresso.exe
JAR Espresso Location: .\tools\espresso\espresso.exe
```

Standard BLIF format (reduced ESPRESSO) - Berkeley Logic Interchange Format

18. Generate ->Generate misII


Program Encryption Toolkit
File Edit BENCH Transform Build Components Variations Reductions Libraries Random Ciruits CIRCLIB Help
Stats Visual Function Form
PLA VHDL UW BLIF misII
c17c17adder [BENCH] c17c17adder[COMPARE-BDD] adder-full [BENCH]
Console
Form
misII

```
-----------------------------------------------
BENCH Output
-----------------------------------------------
# INPUTS
INPUT(1)
INPUT(2)
INPUT(3)

# OUTPUTS
OUTPUT(15)
OUTPUT(8)

# GATES
4 = NOT(1)
5 = AND(1,2)
6 = AND(1,3)
7 = AND(2,3)
8 = OR(6,7,5)
9 = OR(8,4)
10 = NOT(2)
11 = OR(8,10)
12 = NOT(3)
13 = OR(8,12)
14 = NAND(1,2,3)
15 = NAND(13,14,9,11)


-----------------------------------------------
Factors
-----------------------------------------------
[237] = i1'
[20] = i1 i2
[19] = i1 i3
[18] = i2 i3
{o5} = [18] + [19] + [20]
[279] = {o5} + [237]
[238] = i2'
[277] = {o5} + [238]
[236] = i3'
[275] = {o5} + [236]
[273] = i1' + i2' + i3'
{o4} = [273]' + [275]' + [277]' + [279]'


-----------------------------------------------
Technology Map
-----------------------------------------------
[237]       inv1       2.00
```

```
#
# 3 inputs
# 2 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 9
# Intermediate nodes: 9
#    ANDs: 0
#    ORs: 0
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(11)
OUTPUT(12)

4=NOR(2,3)
5=NOR(4,2)
6=NOR(3,4)
7=NOR(5,6)
8=NOR(1,7)
12=NOR(4,8)
10=NOR(7,8)
9=NOR(1,8)
11=NOR(9,10)
```

```
Maximum Fan-In: 2
Maximum Fan-Out: 3
Average Fan-In: 2.0
Average Fan-Out: 1.6

Callback: [generatemisii]
```

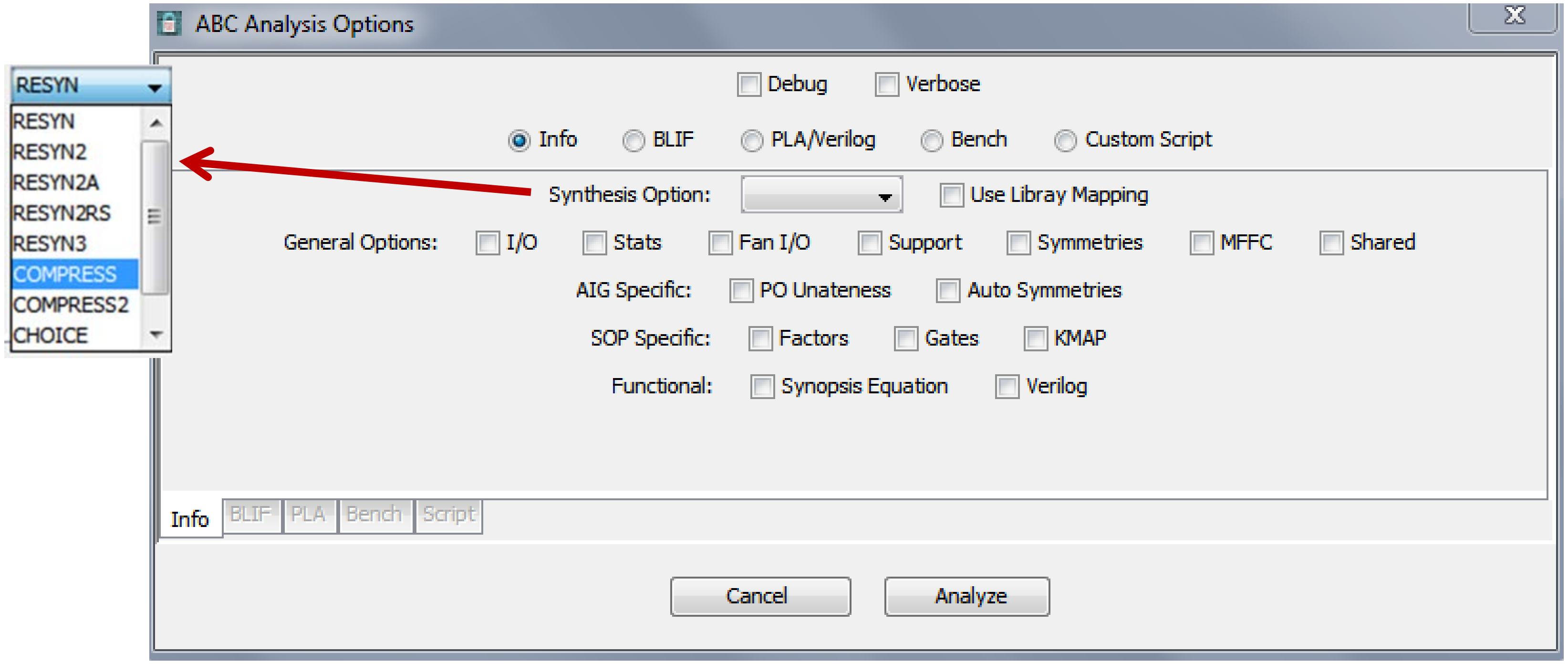
Standard MisII format: Multiple-level Combinational *Logic* Optimization Program

Technology mapping is standard BENCH gates with 2-4 fanin

19. Generate ->Generate ABC

See ABC documentation at: http://people.eecs.berkeley.edu/~alanmi/abc/

Information Options


ABC Analysis Options

Debug | Verbose

[✓ Info] BLIF PLA/Verilog Bench Custom Script

Synthesis Option: | Use Libray Mapping

General Options: I/O Stats Fan I/O Support Symmetries MFFC Shared

AIG Specific: PO Unateness Auto Symmetries

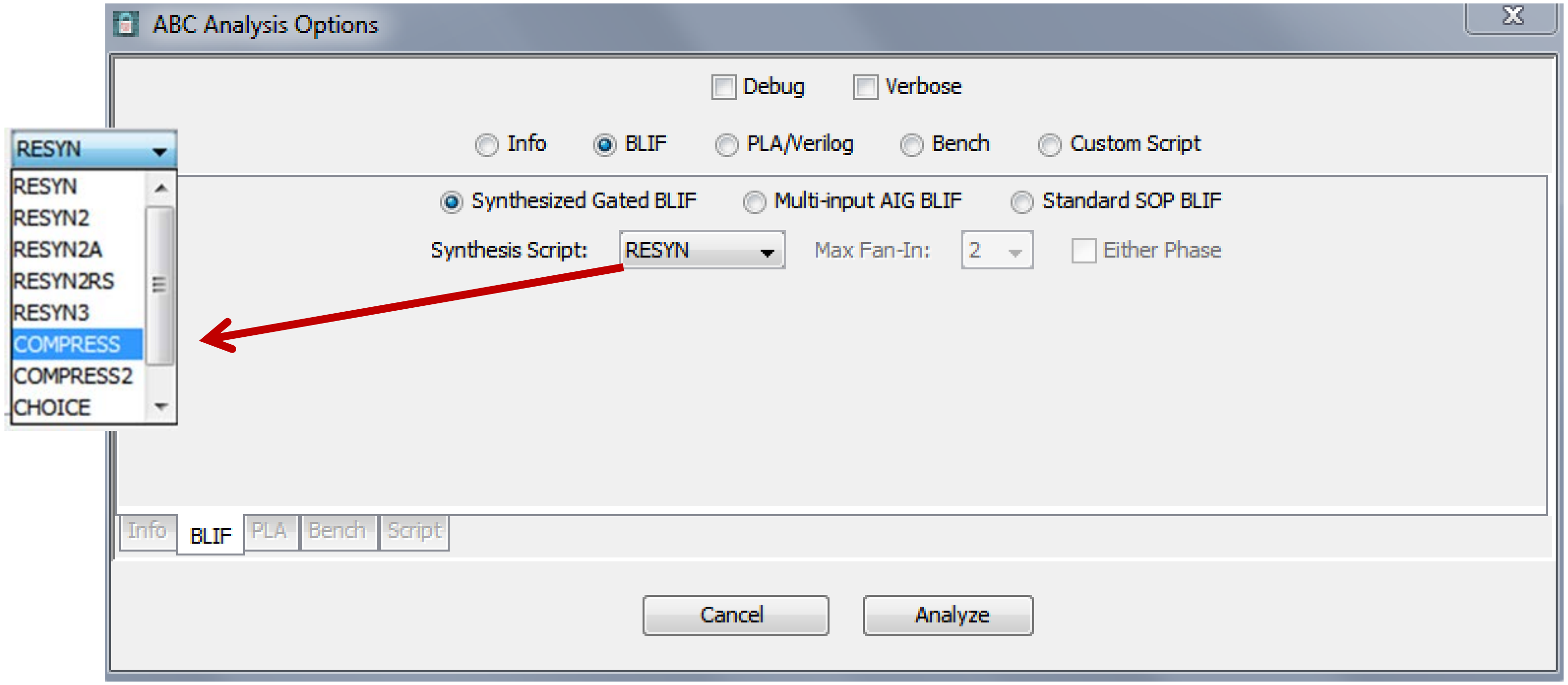
SOP Specific: Factors Gates KMAP

Functional: Synopsis Equation Verilog

19. Generate ->Generate ABC

See ABC documentation at: http://people.eecs.berkeley.edu/~alanmi/abc/

BLIF Synthesis Options


ABC Analysis Options

[ ] Debug [ ] Verbose

Info [✓ BLIF] PLA/Verilog Bench Custom Script

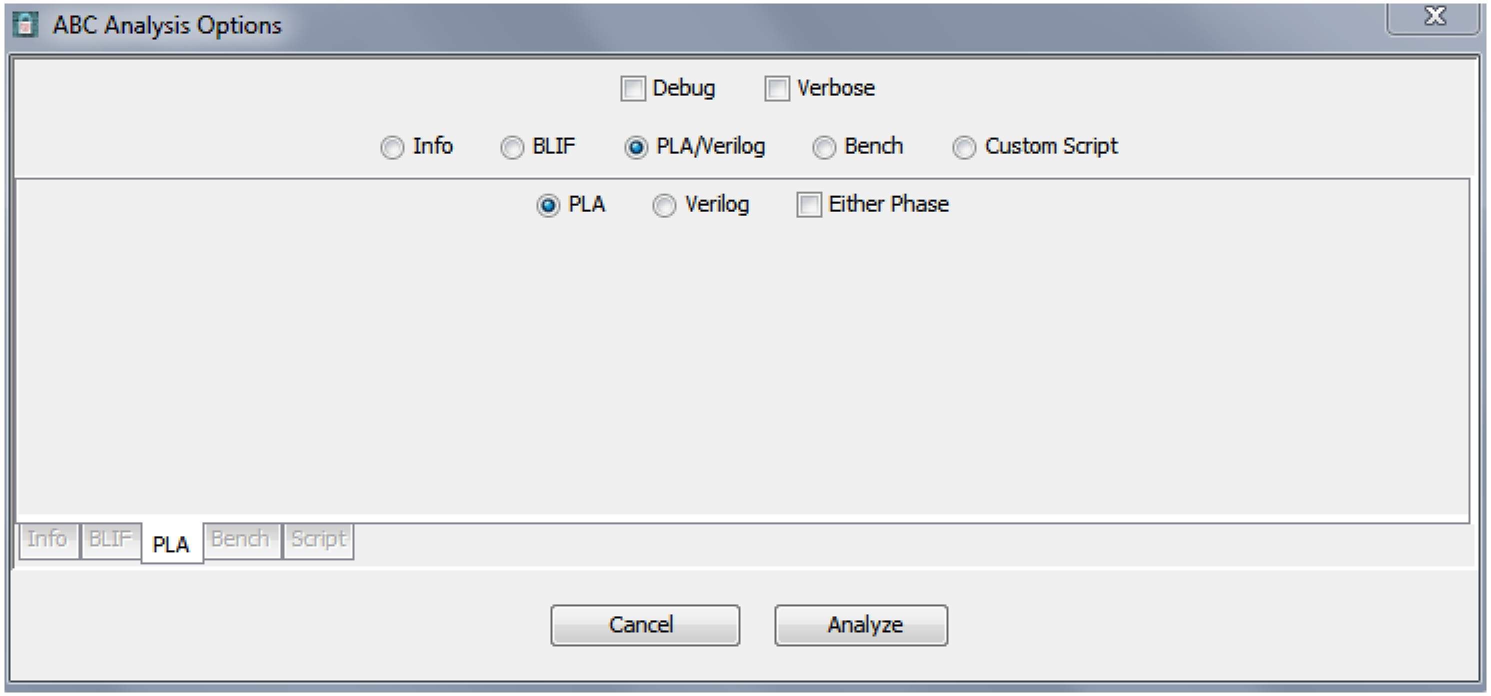
[✓ Synthesized Gated BLIF] Multi-input AIG BLIF Standard SOP BLIF

Synthesis Script: RESYN Max Fan-In: 2 [ ] Either Phase

19. Generate ->Generate ABC

See ABC documentation at: http://people.eecs.berkeley.edu/~alanmi/abc/

PLA / Verilog Options


ABC Analysis Options

[ ] Debug [ ] Verbose

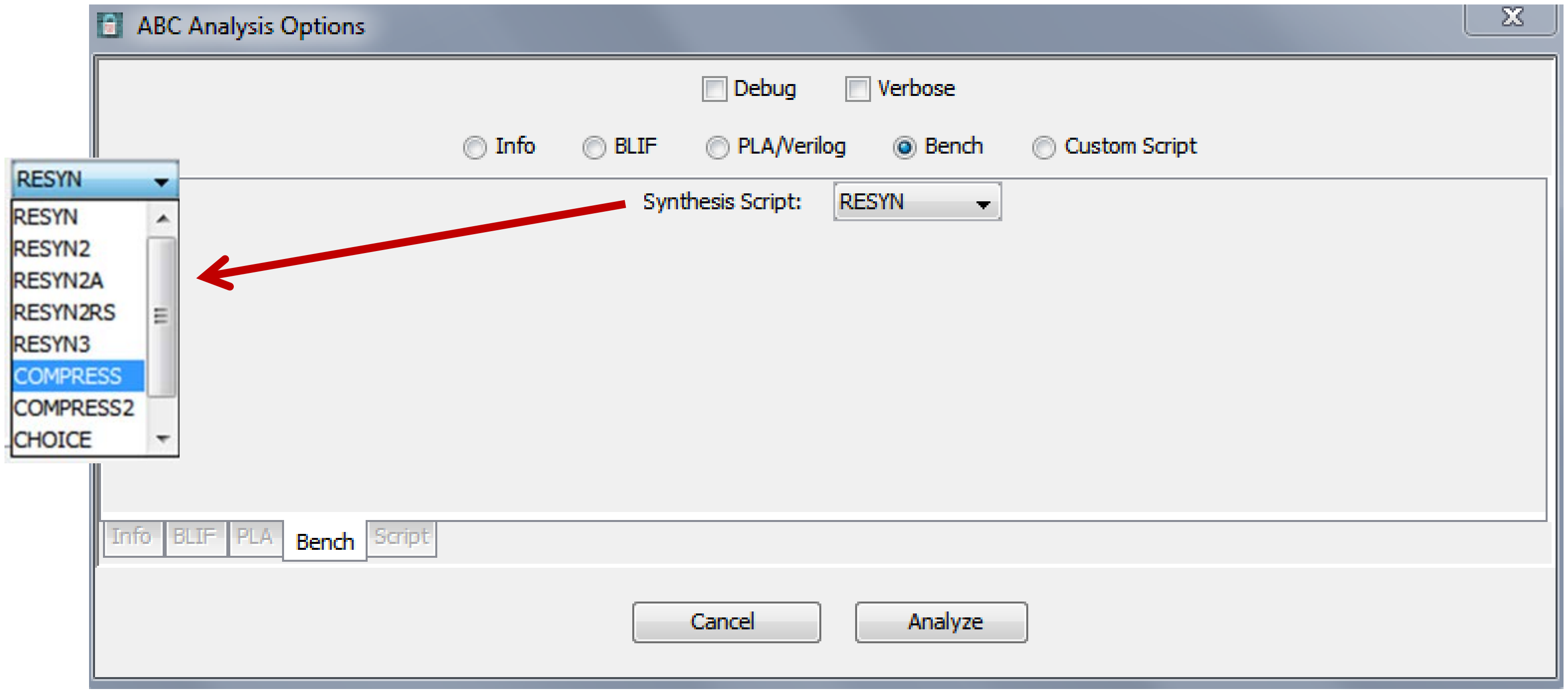
○ Info ○ BLIF [✓ PLA/Verilog] ○ Bench ○ Custom Script

[✓ PLA] ○ Verilog [ ] Either Phase

19. Generate ->Generate ABC

See ABC documentation at: http://people.eecs.berkeley.edu/~alanmi/abc/

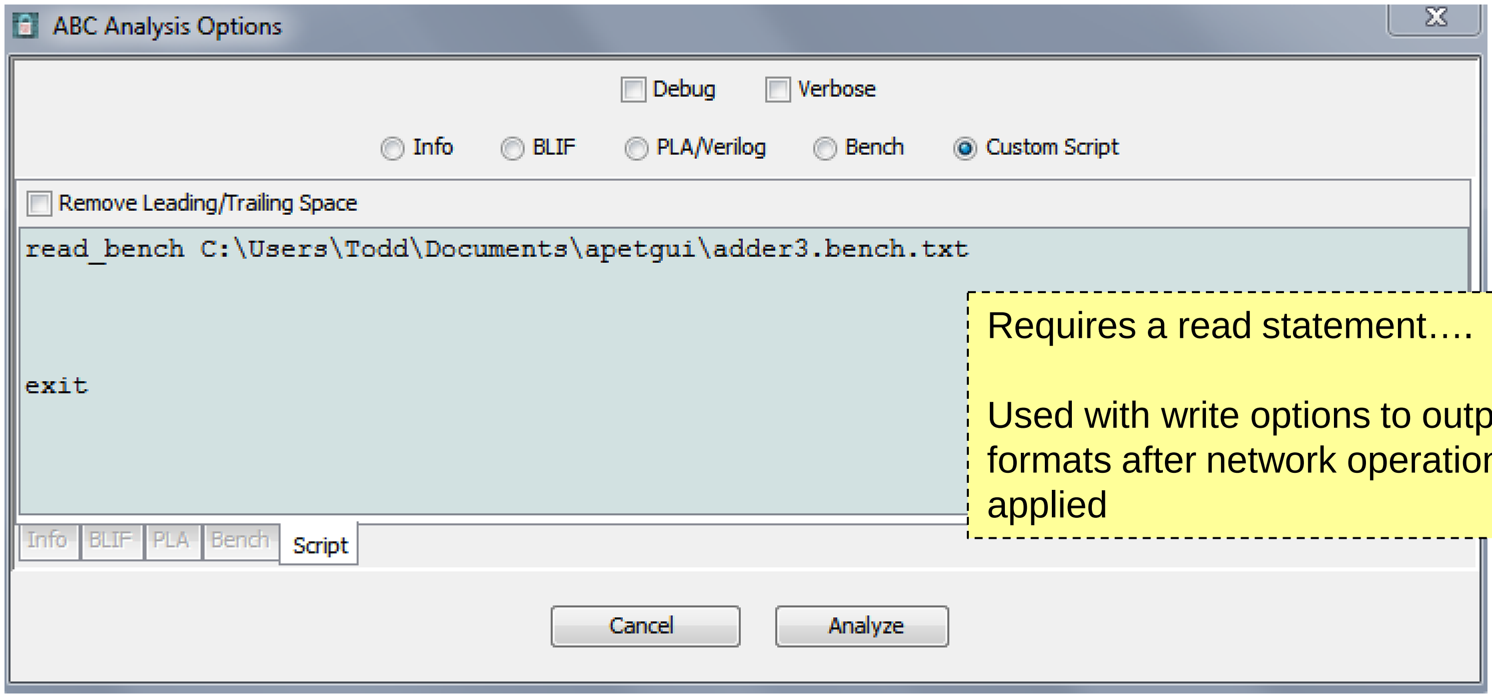
BENCH Generation Options



19. Generate ->Generate ABC

See ABC documentation at: http://people.eecs.berkeley.edu/~alanmi/abc/

Custom Script

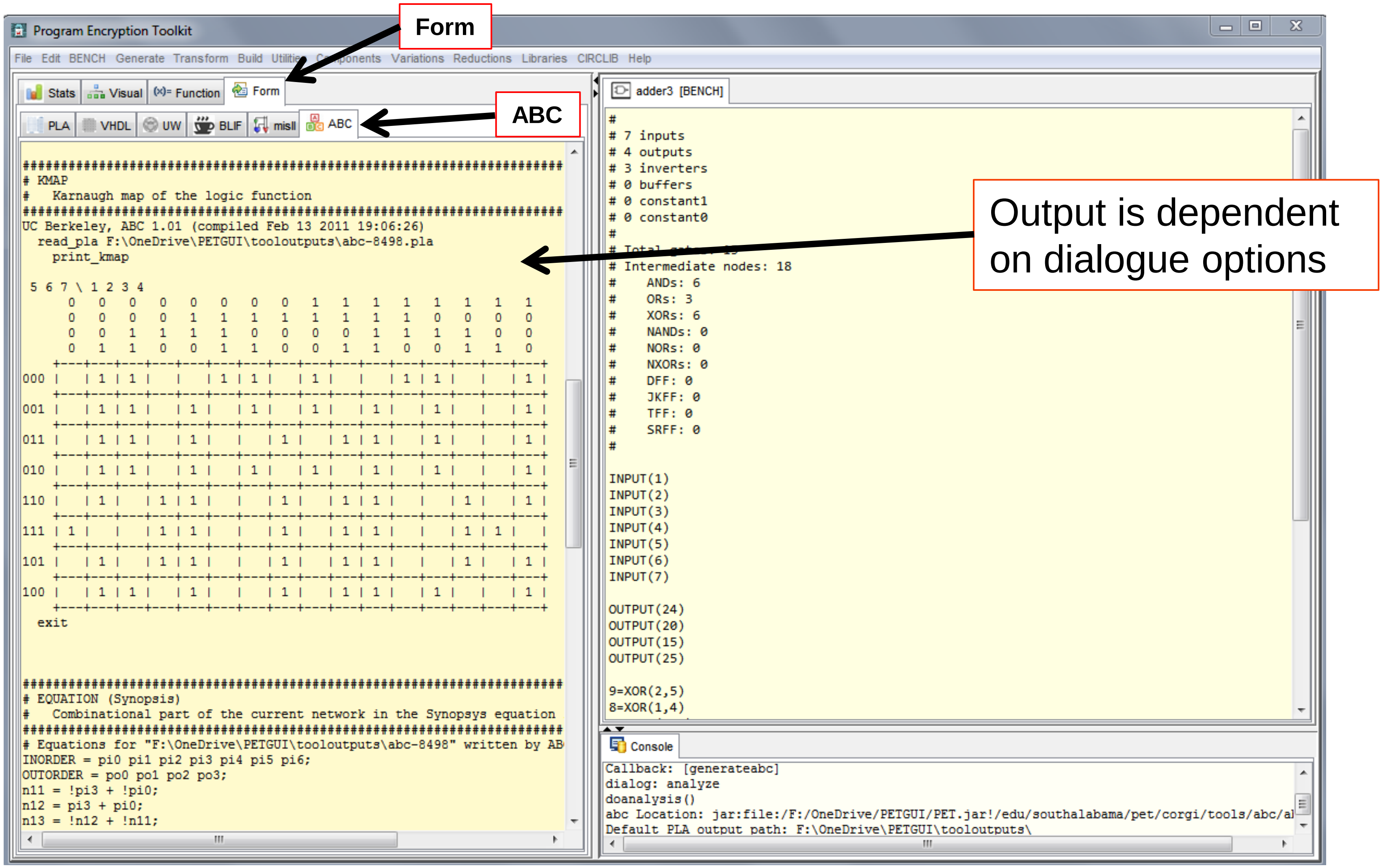


```
read_bench C:\Users\Todd\Documents\apetgui\adder3.bench.txt



exit
```

Requires a read statement….

Used with write options to output file formats after network operations are applied

19. Generate ->Generate ABC

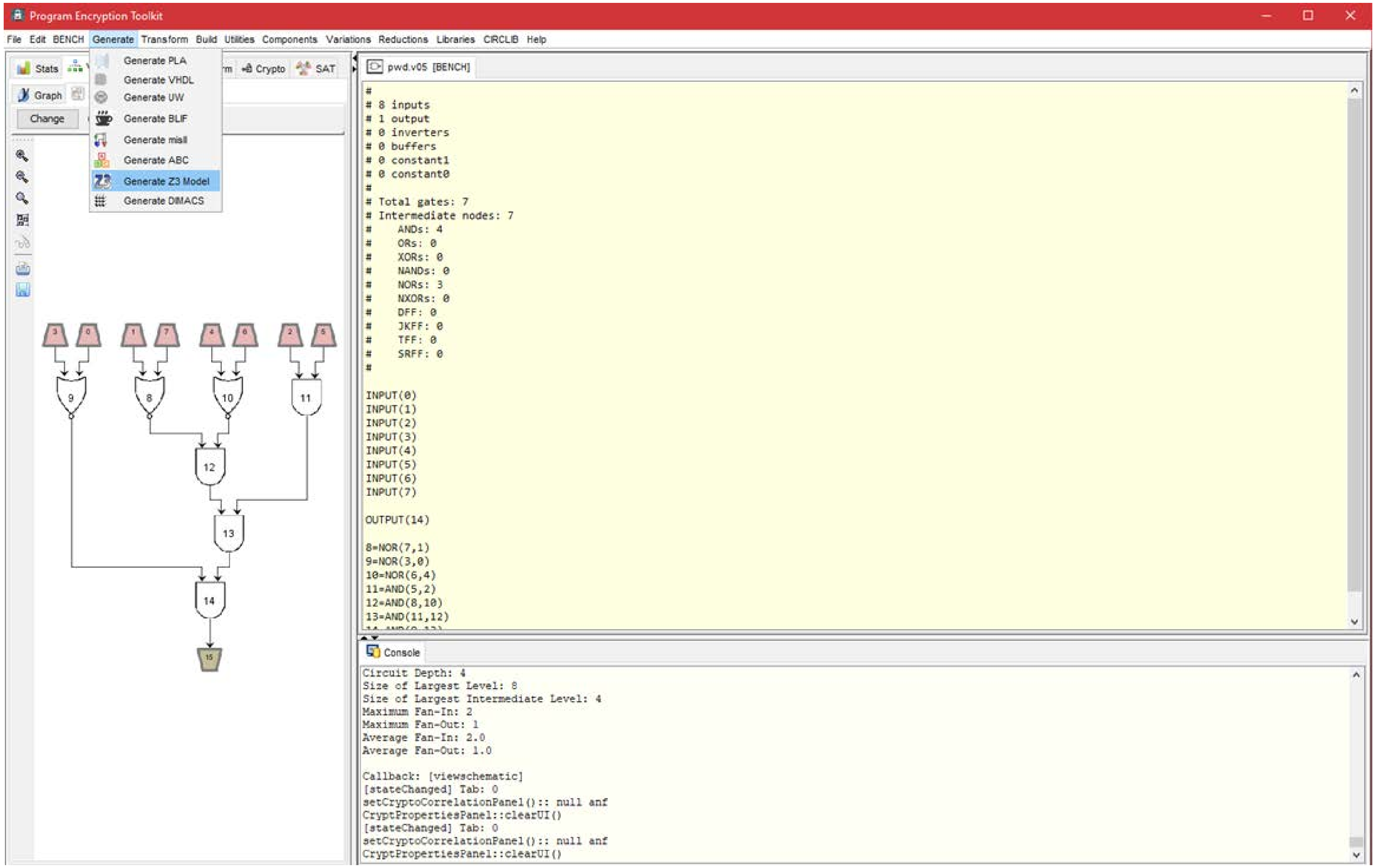

Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form
PLA VHDL UW BLIF misII ABC
adder3 [BENCH]
Console
Form
ABC
Output is dependent on dialogue options

```
########################################################################
# KMAP
#   Karnaugh map of the logic function
########################################################################
UC Berkeley, ABC 1.01 (compiled Feb 13 2011 19:06:26)
  read_pla F:\OneDrive\PETGUI\tooloutputs\abc-8498.pla
    print_kmap

 5 6 7 \ 1 2 3 4
      0   0   0   0   0   0   0   0   1   1   1   1   1   1   1   1
      0   0   0   0   1   1   1   1   1   1   1   1   0   0   0   0
      0   0   1   1   1   1   0   0   0   0   1   1   1   1   0   0
      0   1   1   0   0   1   1   0   0   1   1   0   0   1   1   0
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
000 |   | 1 | 1 |   |   | 1 | 1 |   | 1 |   |   | 1 | 1 |   |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
001 |   | 1 | 1 |   | 1 |   | 1 |   | 1 |   | 1 |   | 1 |   |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
011 |   | 1 | 1 |   | 1 |   |   | 1 |   | 1 | 1 |   | 1 |   |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
010 |   | 1 | 1 |   | 1 |   | 1 |   | 1 |   | 1 |   | 1 |   |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
110 |   | 1 |   | 1 | 1 |   |   | 1 |   | 1 | 1 |   |   | 1 |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
111 | 1 |   |   | 1 | 1 |   |   | 1 |   | 1 | 1 |   |   | 1 | 1 |   |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
101 |   | 1 |   | 1 | 1 |   |   | 1 |   | 1 | 1 |   |   | 1 |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
100 |   | 1 | 1 |   | 1 |   |   | 1 |   | 1 | 1 |   | 1 |   |   | 1 |
    +---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
  exit



########################################################################
# EQUATION (Synopsis)
#   Combinational part of the current network in the Synopsys equation
########################################################################
# Equations for "F:\OneDrive\PETGUI\tooloutputs\abc-8498" written by AB
INORDER = pi0 pi1 pi2 pi3 pi4 pi5 pi6;
OUTORDER = po0 po1 po2 po3;
n11 = !pi3 + !pi0;
n12 = pi3 + pi0;
n13 = !n12 + !n11;
```

```
#
# 7 inputs
# 4 outputs
# 3 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 15
# Intermediate nodes: 18
#     ANDs: 6
#     ORs: 3
#     XORs: 6
#     NANDs: 0
#     NORs: 0
#     NXORs: 0
#     DFF: 0
#     JKFF: 0
#     TFF: 0
#     SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(24)
OUTPUT(20)
OUTPUT(15)
OUTPUT(25)

9=XOR(2,5)
8=XOR(1,4)
```

```
Callback: [generateabc]
dialog: analyze
doanalysis()
abc Location: jar:file:/F:/OneDrive/PETGUI/PET.jar!/edu/southalabama/pet/corgi/tools/abc/al
Default PLA output path: F:\OneDrive\PETGUI\tooloutputs\
```

20. Generate ->Generate z3

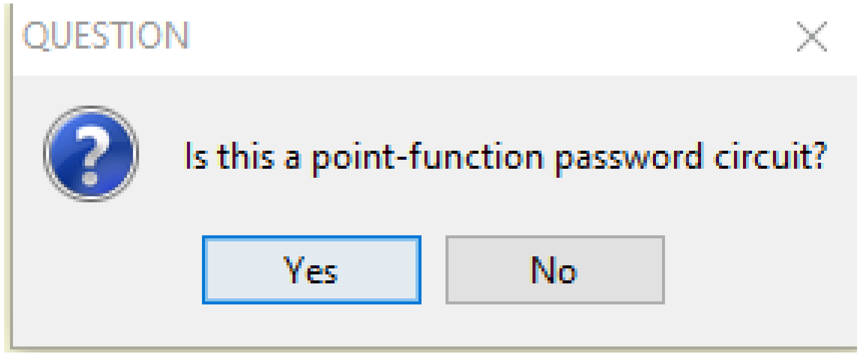


```
#
# 8 inputs
# 1 output
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 7
# Intermediate nodes: 7
#    ANDs: 4
#    ORs: 0
#    XORs: 0
#    NANDs: 0
#    NORs: 3
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(14)

8=NOR(7,1)
9=NOR(3,0)
10=NOR(6,4)
11=AND(5,2)
12=AND(8,10)
13=AND(11,12)
```

```
Circuit Depth: 4
Size of Largest Level: 8
Size of Largest Intermediate Level: 4
Maximum Fan-In: 2
Maximum Fan-Out: 1
Average Fan-In: 2.0
Average Fan-Out: 1.0

Callback: [viewschematic]
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
```

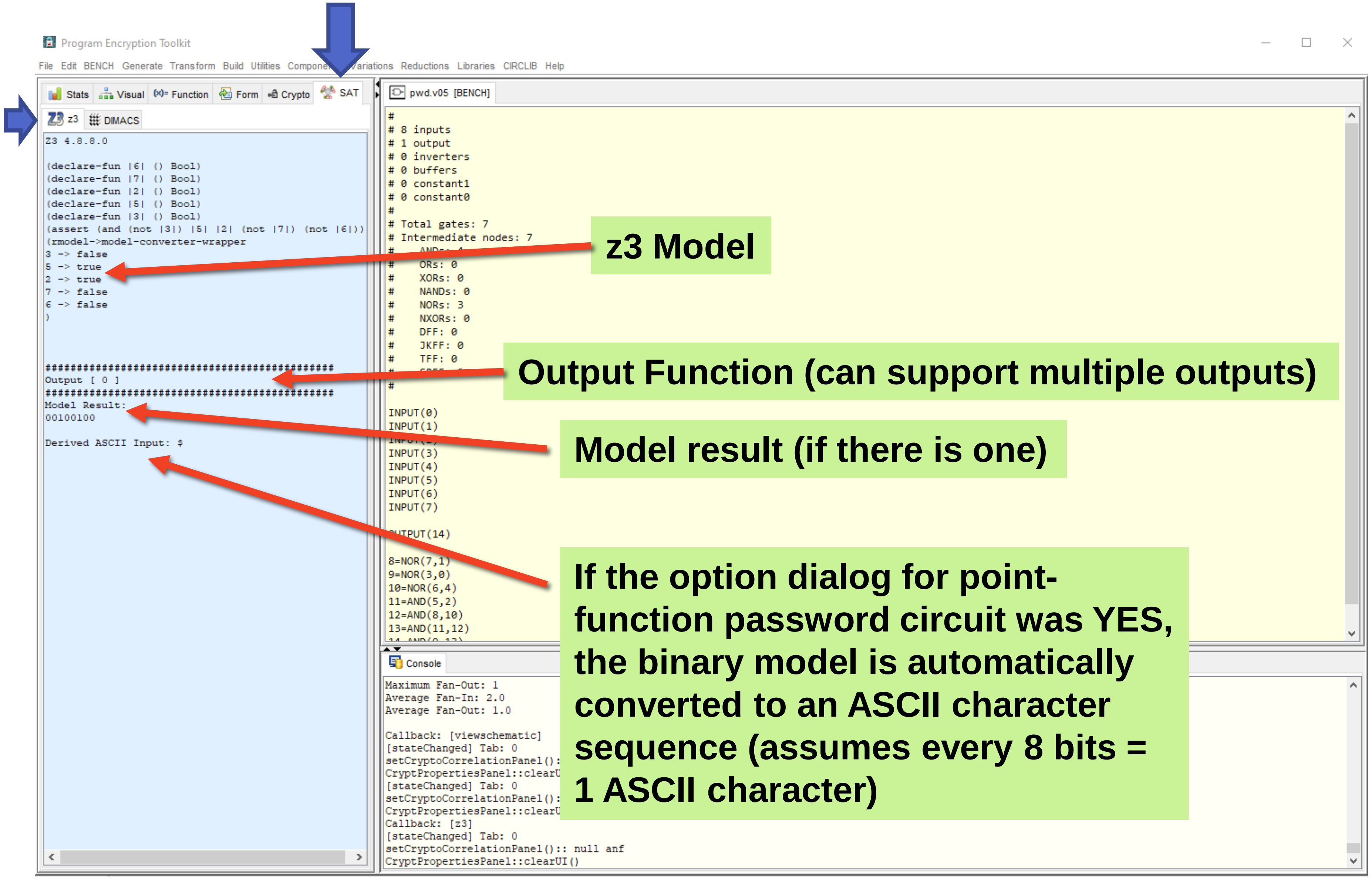
Use Case: Loading and Analyzing BENCH File

- Z3 is a SAT solver and generating a model implies that the circuit will be transformed into a form in which the solver can use
- Finding a model means that it will attempt to find an assignment of values (to the input variables) that will produce a true (1) output
- The solver is geared currently to analyze point-function circuits that represent password-checking functions
  - If you know the circuit takes as input an actual ASCII character sequence and produces a single output (true if password matches, false others), then choose Yes
  - Otherwise, choose No, z3 will produce a model for any circuit, whether it is a point-function circuit and whether or not the input is assumed to be ASCII character sequences



Is this a point-function password circuit?

Use Case: Loading and Analyzing BENCH File


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
z3 DIMACS
pwd.v05 [BENCH]
Console

```
Z3 4.8.8.0

(declare-fun |6| () Bool)
(declare-fun |7| () Bool)
(declare-fun |2| () Bool)
(declare-fun |5| () Bool)
(declare-fun |3| () Bool)
(assert (and (not |3|) |5| |2| (not |7|) (not |6|))
(rmodel->model-converter-wrapper
3 -> false
5 -> true
2 -> true
7 -> false
6 -> false
)


##############################################
Output [ 0 ]
##############################################
Model Result:
00100100

Derived ASCII Input: $
```

```
#
# 8 inputs
# 1 output
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 7
# Intermediate nodes: 7
#    ANDs: 4
#    ORs: 0
#    XORs: 0
#    NANDs: 0
#    NORs: 3
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(14)

8=NOR(7,1)
9=NOR(3,0)
10=NOR(6,4)
11=AND(5,2)
12=AND(8,10)
13=AND(11,12)
14=AND(9,13)
```

```
Maximum Fan-Out: 1
Average Fan-In: 2.0
Average Fan-Out: 1.0

Callback: [viewschematic]
[stateChanged] Tab: 0
setCryptoCorrelationPanel():
CryptPropertiesPanel::clearU
[stateChanged] Tab: 0
setCryptoCorrelationPanel():
CryptPropertiesPanel::clearU
Callback: [z3]
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
```

z3 Model

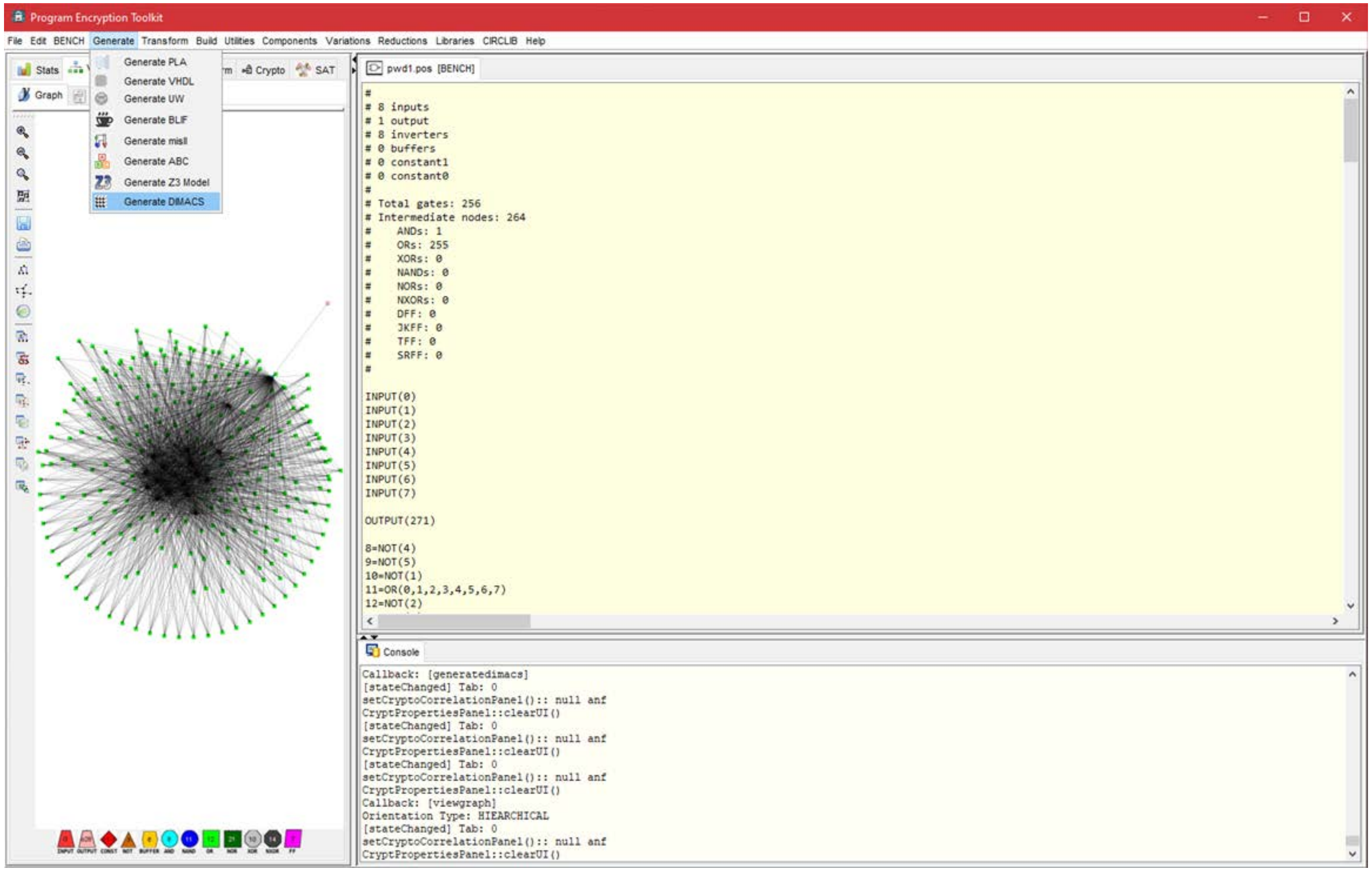
Output Function (can support multiple outputs)

Model result (if there is one)

If the option dialog for point-function password circuit was YES, the binary model is automatically converted to an ASCII character sequence (assumes every 8 bits = 1 ASCII character)

21. Generate ->Generate DIMACs

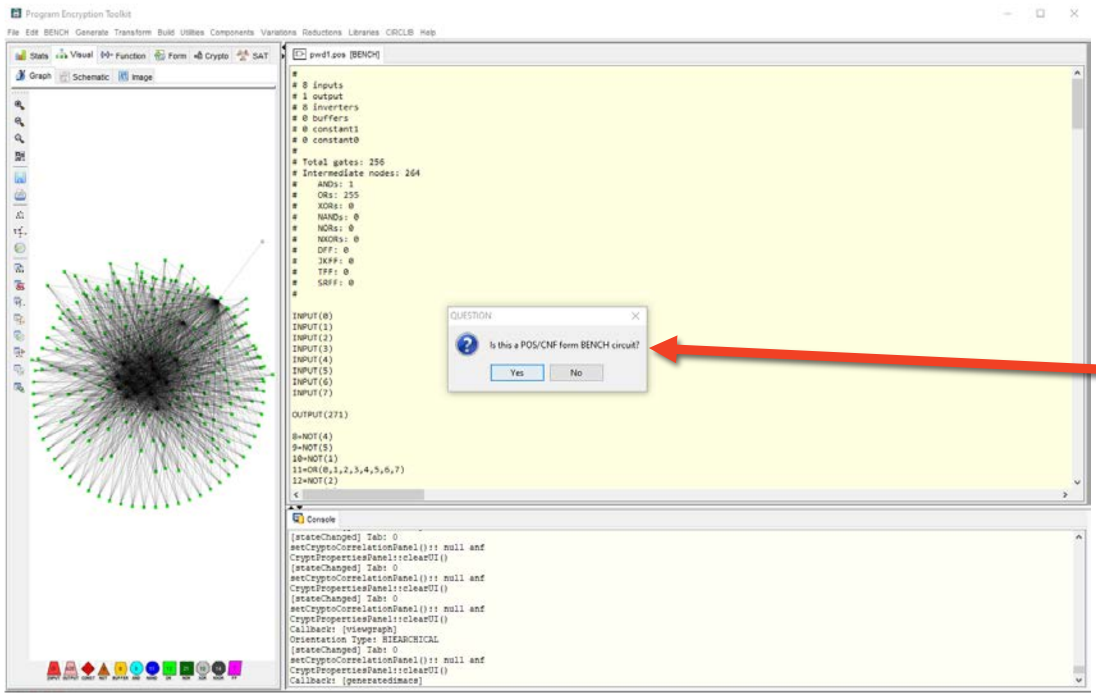
- **DIMACs format is a standard representation method for CNF/POS formulas**
- **Any POS/POM form circuit can be readily translated into a DIMACS format**
- **Only supports SINGLE output functions**



```
#
# 8 inputs
# 1 output
# 8 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 256
# Intermediate nodes: 264
#    ANDs: 1
#    ORs: 255
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(271)

8=NOT(4)
9=NOT(5)
10=NOT(1)
11=OR(0,1,2,3,4,5,6,7)
12=NOT(2)
```

```
Callback: [generatedimacs]
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [viewgraph]
Orientation Type: HIEARCHICAL
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
```

21. Generate ->Generate DIMACs

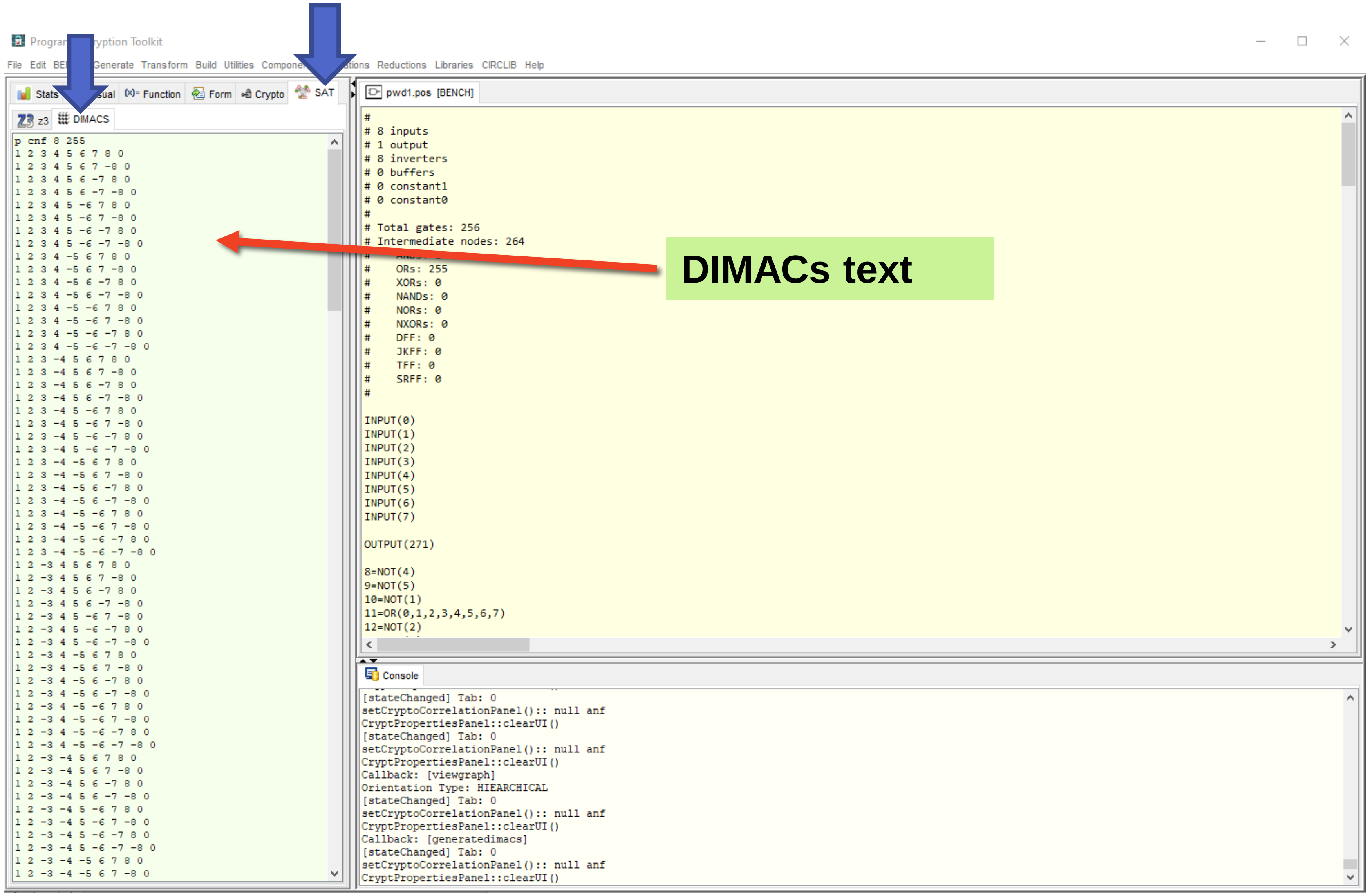

Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
Graph Schematic Image
pwd1.pos [BENCH]
QUESTION
Is this a POS/CNF form BENCH circuit?
Yes
No
Console

```
#
# 8 inputs
# 1 output
# 8 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 256
# Intermediate nodes: 264
#    ANDs: 1
#    ORs: 255
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(271)

8=NOT(4)
9=NOT(5)
10=NOT(1)
11=OR(0,1,2,3,4,5,6,7)
12=NOT(2)
```

```
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [viewgraph]
Orientation Type: HIEARCHICAL
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [generatedimacs]
```

If the circuit is synthesized as a Product of Sums/Product of Maxterms structure, choose YES

21. Generate ->Generate DIMACs

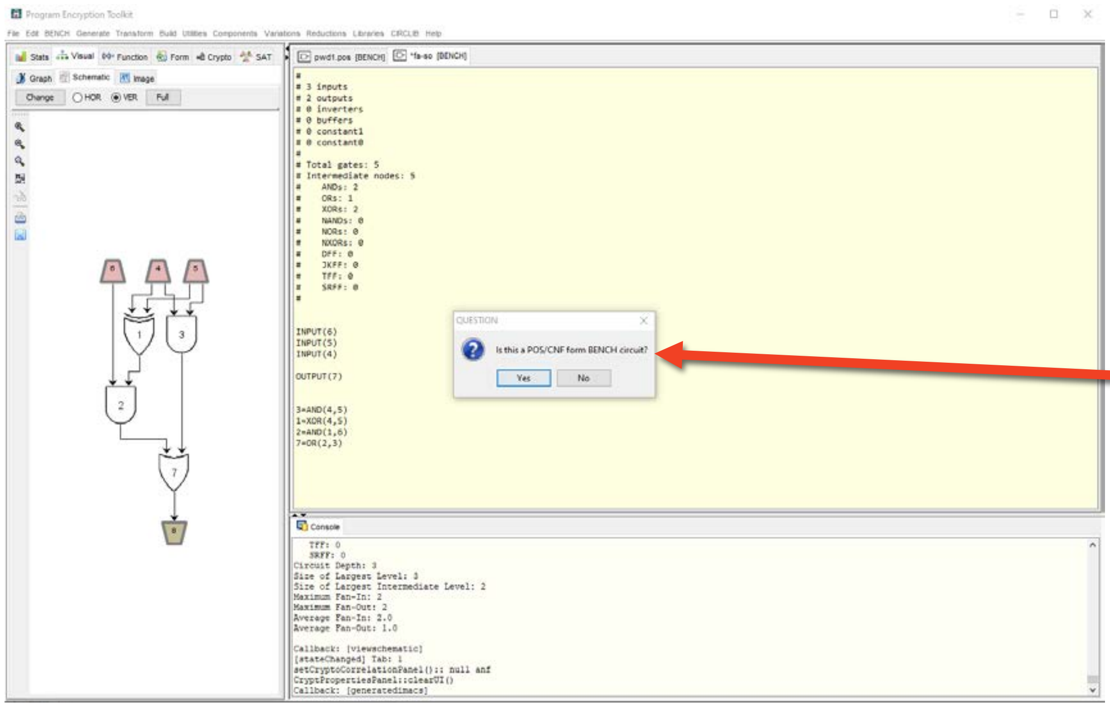

Program... ...ryption Toolkit
File Edit BE... ...Generate Transform Build Utilities Compone... ...ations Reductions Libraries CIRCLIB Help
Stats ...sual Function Form Crypto SAT
z3 DIMACS
pwd1.pos [BENCH]
Console
DIMACs text

```
p cnf 8 255
1 2 3 4 5 6 7 8 0
1 2 3 4 5 6 7 -8 0
1 2 3 4 5 6 -7 8 0
1 2 3 4 5 6 -7 -8 0
1 2 3 4 5 -6 7 8 0
1 2 3 4 5 -6 7 -8 0
1 2 3 4 5 -6 -7 8 0
1 2 3 4 5 -6 -7 -8 0
1 2 3 4 -5 6 7 8 0
1 2 3 4 -5 6 7 -8 0
1 2 3 4 -5 6 -7 8 0
1 2 3 4 -5 6 -7 -8 0
1 2 3 4 -5 -6 7 8 0
1 2 3 4 -5 -6 7 -8 0
1 2 3 4 -5 -6 -7 8 0
1 2 3 4 -5 -6 -7 -8 0
1 2 3 -4 5 6 7 8 0
1 2 3 -4 5 6 7 -8 0
1 2 3 -4 5 6 -7 8 0
1 2 3 -4 5 6 -7 -8 0
1 2 3 -4 5 -6 7 8 0
1 2 3 -4 5 -6 7 -8 0
1 2 3 -4 5 -6 -7 8 0
1 2 3 -4 5 -6 -7 -8 0
1 2 3 -4 -5 6 7 8 0
1 2 3 -4 -5 6 7 -8 0
1 2 3 -4 -5 6 -7 8 0
1 2 3 -4 -5 6 -7 -8 0
1 2 3 -4 -5 -6 7 8 0
1 2 3 -4 -5 -6 7 -8 0
1 2 3 -4 -5 -6 -7 8 0
1 2 3 -4 -5 -6 -7 -8 0
1 2 -3 4 5 6 7 8 0
1 2 -3 4 5 6 7 -8 0
1 2 -3 4 5 6 -7 8 0
1 2 -3 4 5 6 -7 -8 0
1 2 -3 4 5 -6 7 -8 0
1 2 -3 4 5 -6 -7 8 0
1 2 -3 4 5 -6 -7 -8 0
1 2 -3 4 -5 6 7 8 0
1 2 -3 4 -5 6 7 -8 0
1 2 -3 4 -5 6 -7 8 0
1 2 -3 4 -5 6 -7 -8 0
1 2 -3 4 -5 -6 7 8 0
1 2 -3 4 -5 -6 7 -8 0
1 2 -3 4 -5 -6 -7 8 0
1 2 -3 4 -5 -6 -7 -8 0
1 2 -3 -4 5 6 7 8 0
1 2 -3 -4 5 6 7 -8 0
1 2 -3 -4 5 6 -7 8 0
1 2 -3 -4 5 6 -7 -8 0
1 2 -3 -4 5 -6 7 8 0
1 2 -3 -4 5 -6 7 -8 0
1 2 -3 -4 5 -6 -7 8 0
1 2 -3 -4 5 -6 -7 -8 0
1 2 -3 -4 -5 6 7 8 0
1 2 -3 -4 -5 6 7 -8 0
```

```
#
# 8 inputs
# 1 output
# 8 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 256
# Intermediate nodes: 264
#    ANDs: [illegible]
#    ORs: 255
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(271)

8=NOT(4)
9=NOT(5)
10=NOT(1)
11=OR(0,1,2,3,4,5,6,7)
12=NOT(2)
```

```
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [viewgraph]
Orientation Type: HIEARCHICAL
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [generatedimacs]
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
```

21. Generate ->Generate DIMACs

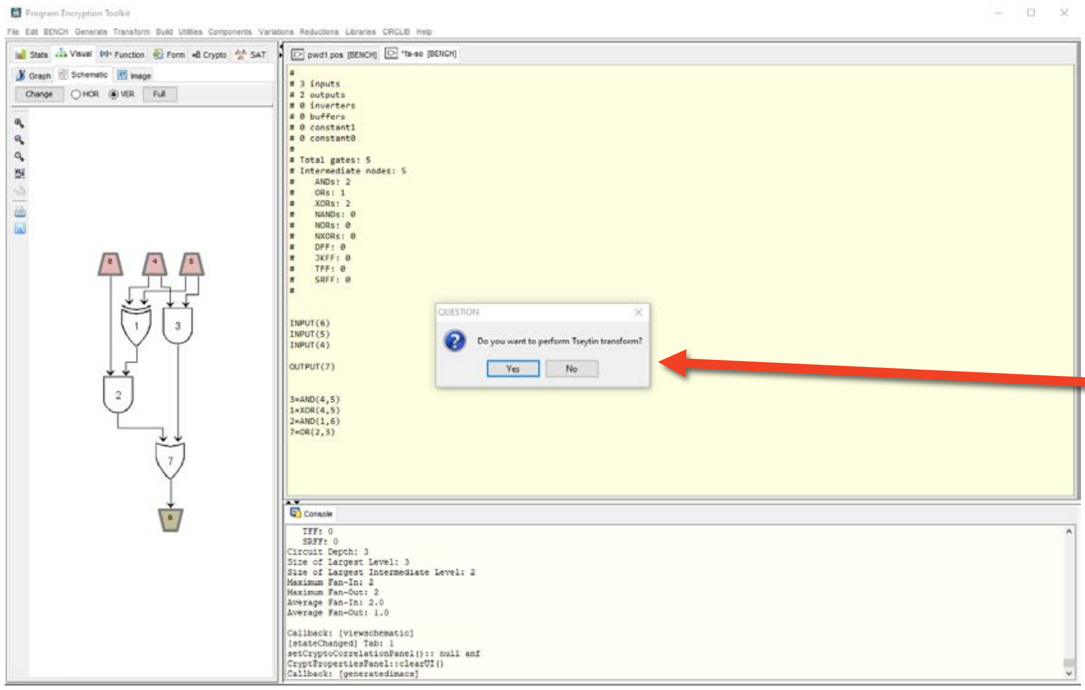

Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
Graph Schematic Image
Change HOR VER Full
pwd1.pos [BENCH]
*fa-so [BENCH]
QUESTION
Is this a POS/CNF form BENCH circuit?
Yes
No
Console

```
#
# 3 inputs
# 2 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 5
# Intermediate nodes: 5
#    ANDs: 2
#    ORs: 1
#    XORs: 2
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#


INPUT(6)
INPUT(5)
INPUT(4)

OUTPUT(7)


3=AND(4,5)
1=XOR(4,5)
2=AND(1,6)
7=OR(2,3)
```

```
   TFF: 0
   SRFF: 0
Circuit Depth: 3
Size of Largest Level: 3
Size of Largest Intermediate Level: 2
Maximum Fan-In: 2
Maximum Fan-Out: 2
Average Fan-In: 2.0
Average Fan-Out: 1.0

Callback: [viewschematic]
[stateChanged] Tab: 1
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [generatedimacs]
```

If the circuit is NOT synthesized as a Product of Sums/Product of Maxterms structure, choose NO

21. Generate ->Generate DIMACs


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
Graph Schematic Image
Change HOR VER Full
pwd1.pos [BENCH] *fa-so [BENCH]
Console

```
#
# 3 inputs
# 2 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 5
# Intermediate nodes: 5
#    ANDs: 2
#    ORs: 1
#    XORs: 2
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#


INPUT(6)
INPUT(5)
INPUT(4)

OUTPUT(7)


3=AND(4,5)
1=XOR(4,5)
2=AND(1,6)
7=OR(2,3)
```

QUESTION

Do you want to perform Tseytin transform?

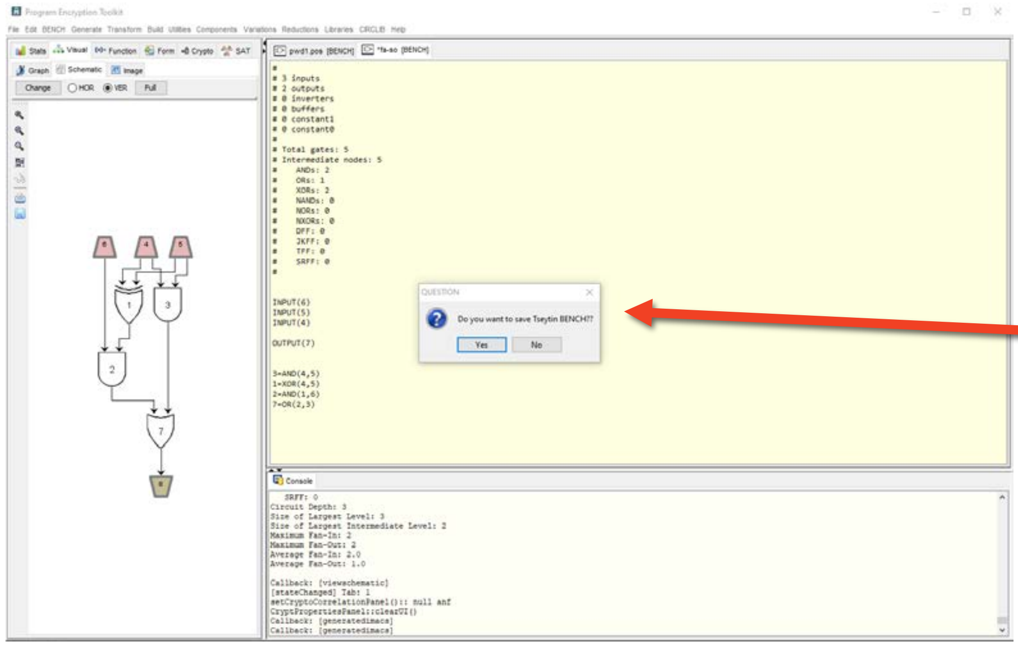
Yes No

```
   TFF: 0
   SRFF: 0
Circuit Depth: 3
Size of Largest Level: 3
Size of Largest Intermediate Level: 2
Maximum Fan-In: 2
Maximum Fan-Out: 2
Average Fan-In: 2.0
Average Fan-Out: 1.0

Callback: [viewschematic]
[stateChanged] Tab: 1
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [generatedimacs]
```

Option is given to transform the circuit via the Tseytin algorithm

1) Choosing NO exits
2) Choosing YES will continue the DIMACs generation

21. Generate ->Generate DIMACs


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLE! Help
Stats Visual Function Form Crypto SAT
Graph Schematic Image
Change HOR VER Full
pwd1.pos [BENCH]
*fa-so [BENCH]
6 4 5 1 3 2 7 8
QUESTION
Yes
No
Console

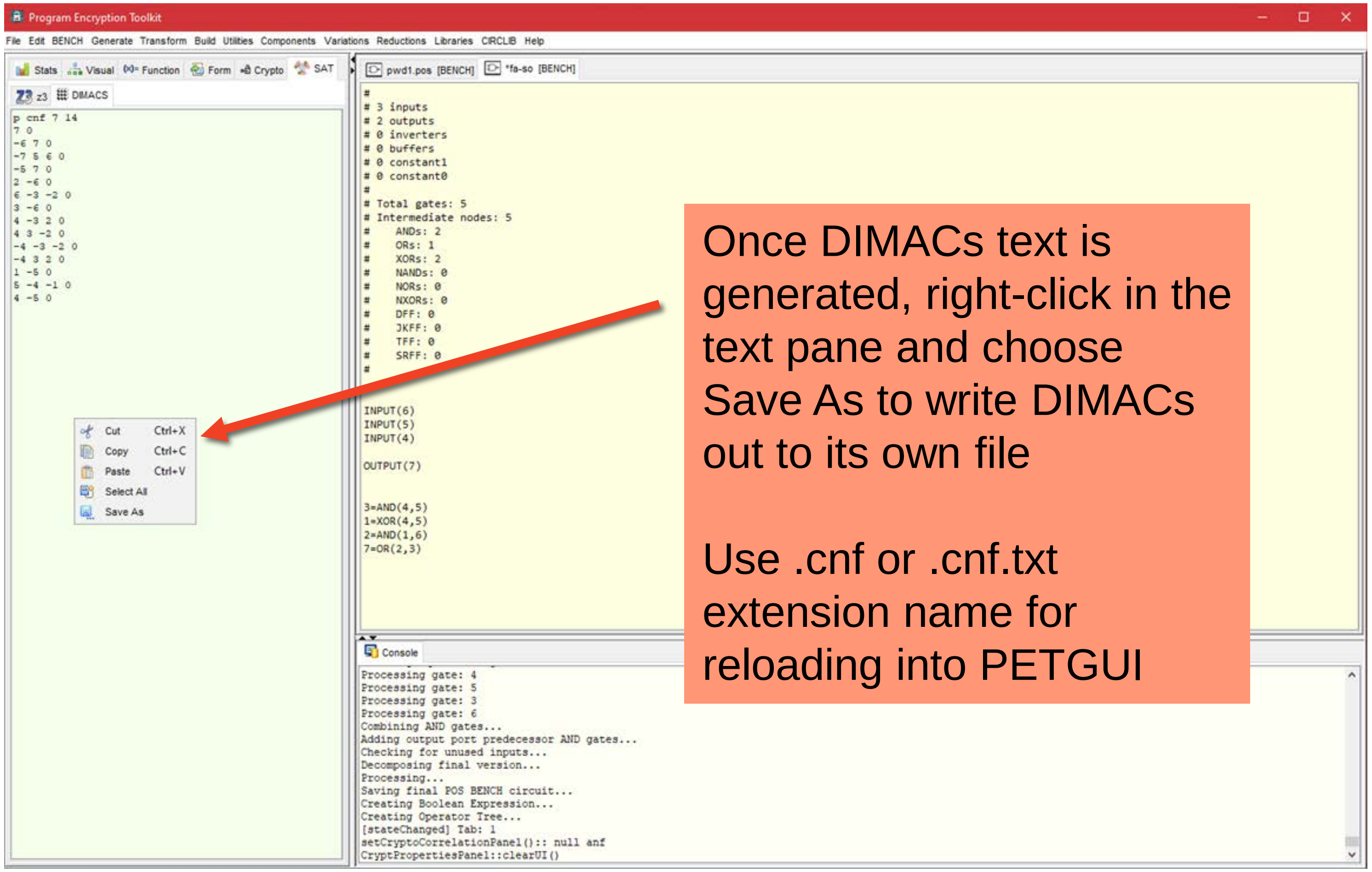
Do you want to save Tseytin BENCH??

```
#
# 3 inputs
# 2 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 5
# Intermediate nodes: 5
#    ANDs: 2
#    ORs: 1
#    XORs: 2
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#


INPUT(6)
INPUT(5)
INPUT(4)

OUTPUT(7)


3=AND(4,5)
1=XOR(4,5)
2=AND(1,6)
7=OR(2,3)
```

```
   SRFF: 0
Circuit Depth: 3
Size of Largest Level: 3
Size of Largest Intermediate Level: 2
Maximum Fan-In: 2
Maximum Fan-Out: 2
Average Fan-In: 2.0
Average Fan-Out: 1.0

Callback: [viewschematic]
[stateChanged] Tab: 1
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
Callback: [generatedimacs]
Callback: [generatedimacs]
```

Next option allows saving of the Tseytin transformed BENCH file to be saved to disk

21. Generate ->Generate DIMACs


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
z3 DIMACS
pwd1.pos [BENCH] *fa-so [BENCH]
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All
Save As
Console

```
p cnf 7 14
7 0
-6 7 0
-7 5 6 0
-5 7 0
2 -6 0
6 -3 -2 0
3 -6 0
4 -3 2 0
4 3 -2 0
-4 -3 -2 0
-4 3 2 0
1 -5 0
5 -4 -1 0
4 -5 0
```

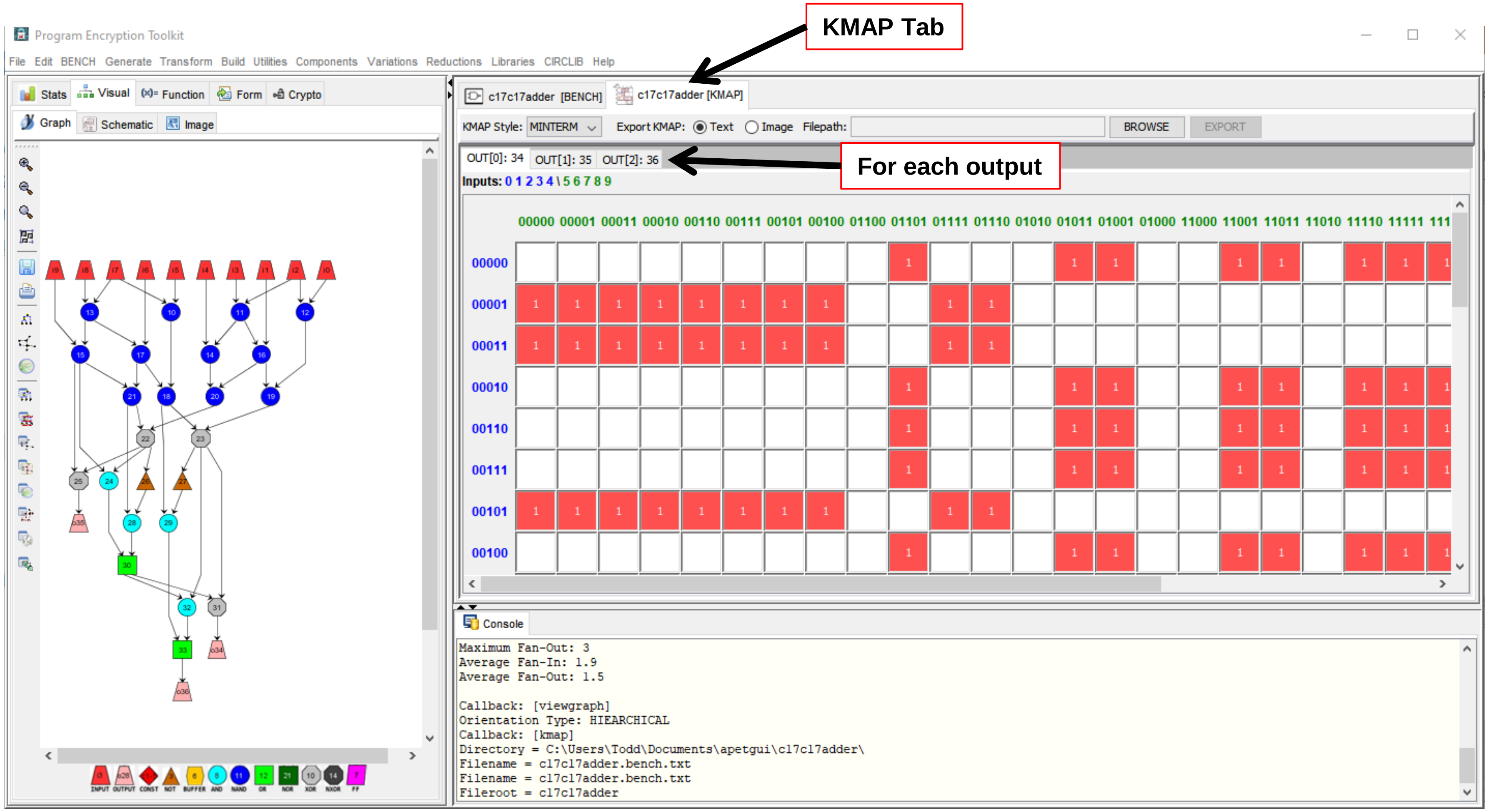
```
#
# 3 inputs
# 2 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 5
# Intermediate nodes: 5
#    ANDs: 2
#    ORs: 1
#    XORs: 2
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#


INPUT(6)
INPUT(5)
INPUT(4)

OUTPUT(7)


3=AND(4,5)
1=XOR(4,5)
2=AND(1,6)
7=OR(2,3)
```

```
Processing gate: 4
Processing gate: 5
Processing gate: 3
Processing gate: 6
Combining AND gates...
Adding output port predecessor AND gates...
Checking for unused inputs...
Decomposing final version...
Processing...
Saving final POS BENCH circuit...
Creating Boolean Expression...
Creating Operator Tree...
[stateChanged] Tab: 1
setCryptoCorrelationPanel():: null anf
CryptPropertiesPanel::clearUI()
```

Once DIMACs text is generated, right-click in the text pane and choose Save As to write DIMACs out to its own file

Use .cnf or .cnf.txt extension name for reloading into PETGUI

22. BENCH ->Karnaugh Map


KMAP Tab
For each output
Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto
Graph Schematic Image
c17c17adder [BENCH]
c17c17adder [KMAP]
BROWSE
EXPORT
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF
Console

KMAP Style: MINTERM Export KMAP: [✓ Text] Image Filepath:

OUT[0]: 34 OUT[1]: 35 OUT[2]: 36

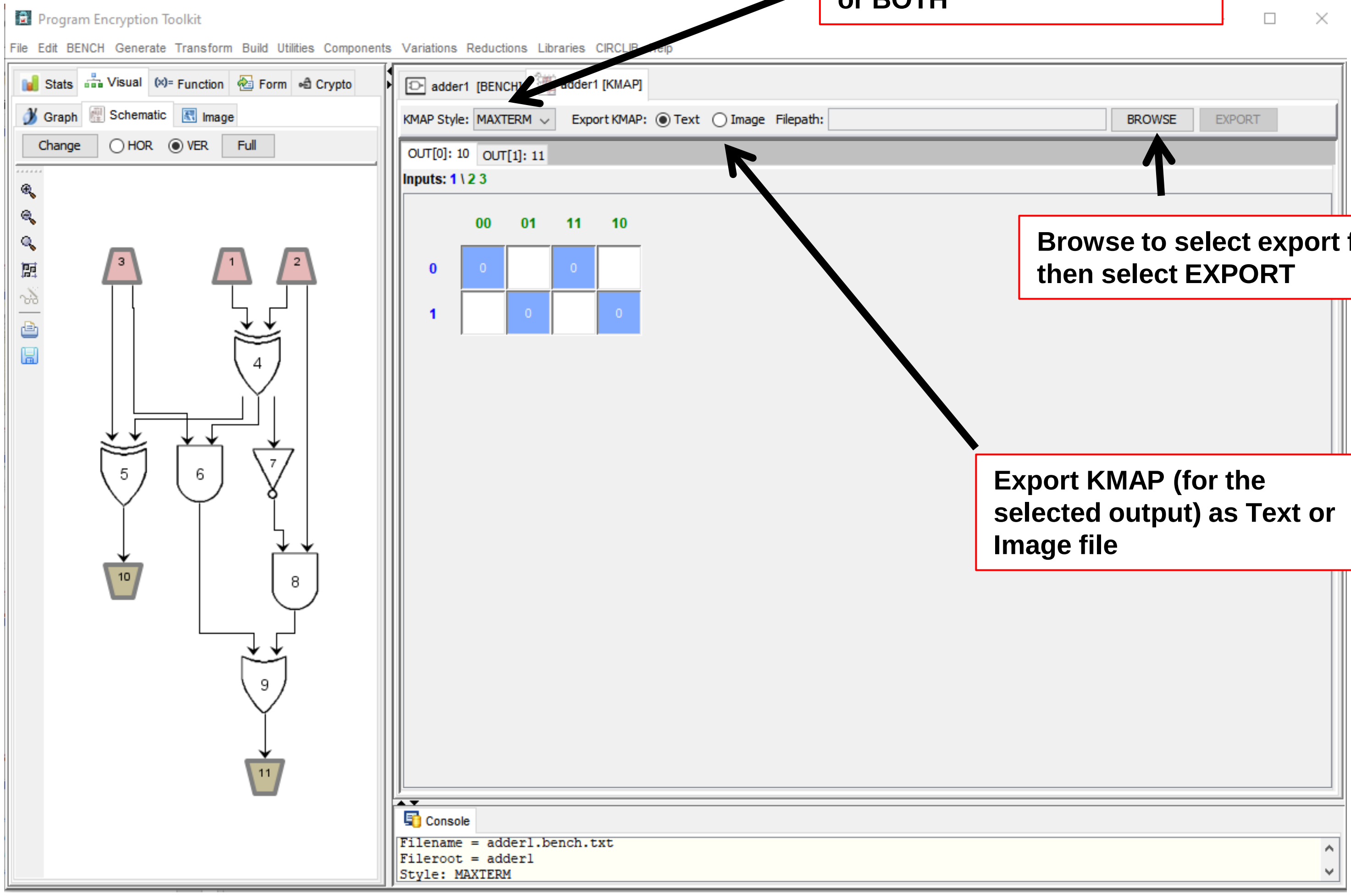
Inputs: 0 1 2 3 4 \ 5 6 7 8 9

	00000	00001	00011	00010	00110	00111	00101	00100	01100	01101	01111	01110	01010	01011	01001	01000	11000	11001	11011	11010	11110	11111	111
00000										1				1	1			1	1		1	1	1
00001	1	1	1	1	1	1	1	1			1	1											
00011	1	1	1	1	1	1	1	1			1	1											
00010										1				1	1			1	1		1	1	1
00110										1				1	1			1	1		1	1	1
00111										1				1	1			1	1		1	1	1
00101	1	1	1	1	1	1	1	1			1	1											
00100										1				1	1			1	1		1	1	1

```
Maximum Fan-Out: 3
Average Fan-In: 1.9
Average Fan-Out: 1.5

Callback: [viewgraph]
Orientation Type: HIEARCHICAL
Callback: [kmap]
Directory = C:\Users\Todd\Documents\apetgui\c17c17adder\
Filename = c17c17adder.bench.txt
Filename = c17c17adder.bench.txt
Fileroot = c17c17adder
```

22. BENCH ->Karnaugh Map

View MINTERM/MAXTERM or BOTH


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto
Graph Schematic Image
Change HOR VER Full
adder1 [BENCH] adder1 [KMAP]
BROWSE
EXPORT
OUT[0]: 10
OUT[1]: 11
3
1
2
4
5
6
7
8
9
10
11
Console

KMAP Style: MAXTERM Export KMAP: [✓ Text] Image Filepath:

Inputs: 1 \ 2 3

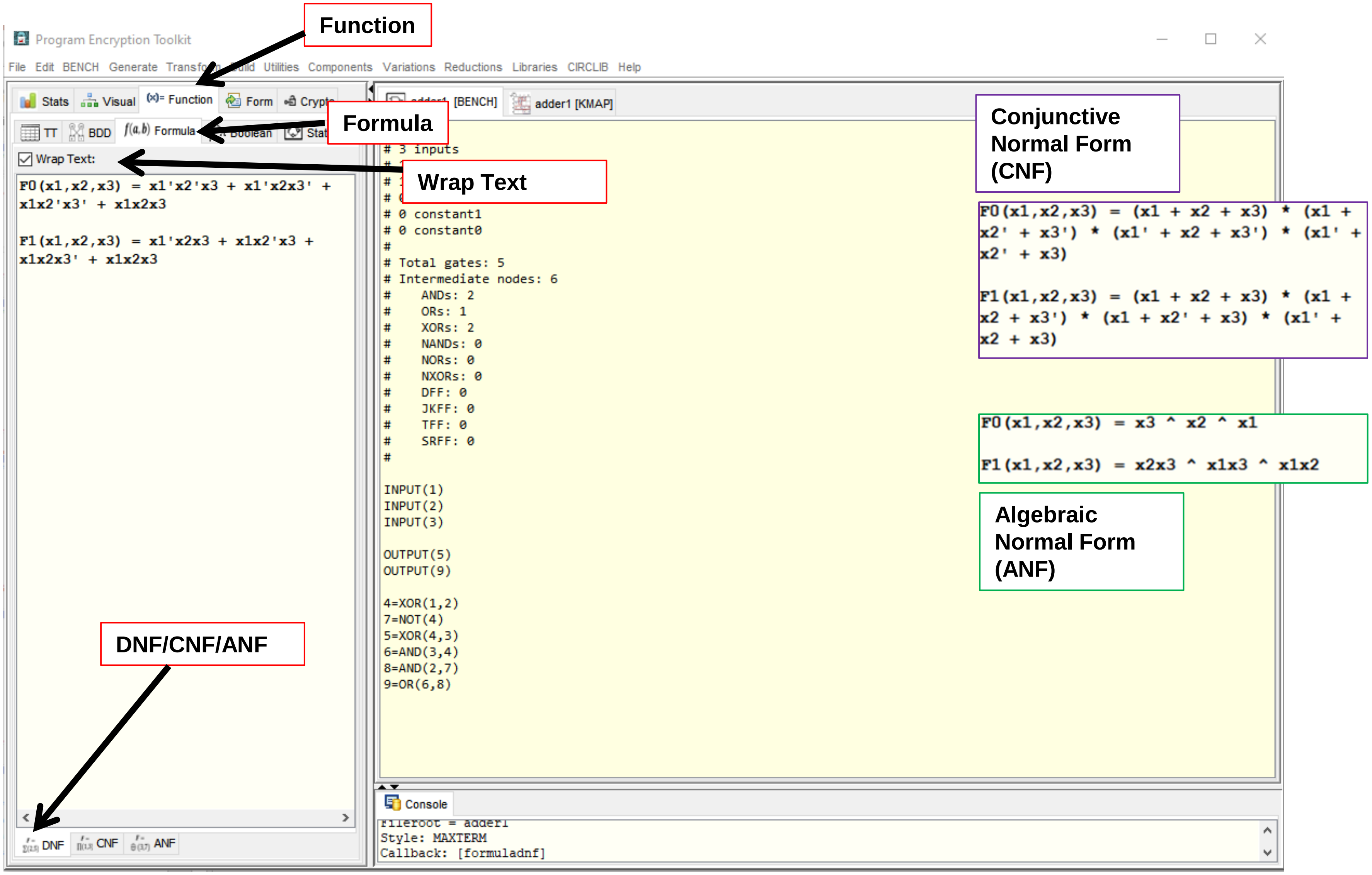
	00	01	11	10
0	0		0	
1		0		0

```
Filename = adder1.bench.txt
Fileroot = adder1
Style: MAXTERM
```

Browse to select export file, then select EXPORT

Export KMAP (for the selected output) as Text or Image file

23. BENCH ->Formula->{DNF,CNF,ANF}


Function
Formula
Wrap Text
DNF/CNF/ANF
Conjunctive Normal Form (CNF)
Algebraic Normal Form (ANF)
Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto
TT BDD Formula Boolean Stat
Wrap Text:
adder1 [BENCH] adder1 [KMAP]
Console
DNF CNF ANF

```
F0(x1,x2,x3) = x1'x2'x3 + x1'x2x3' + x1x2'x3' + x1x2x3

F1(x1,x2,x3) = x1'x2x3 + x1x2'x3 + x1x2x3' + x1x2x3
```

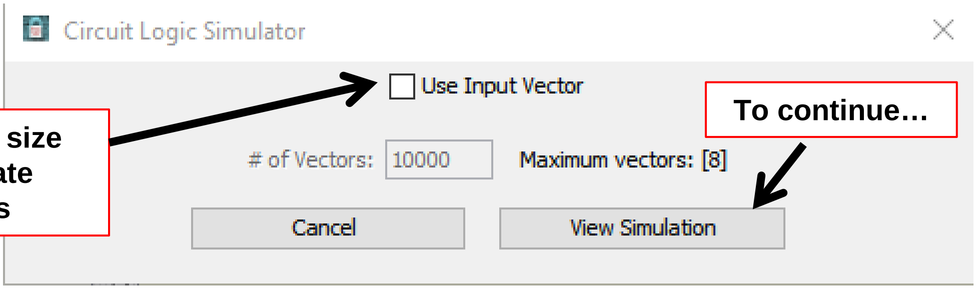
```
# 3 inputs
#
#
#
# 0 constant1
# 0 constant0
#
# Total gates: 5
# Intermediate nodes: 6
#    ANDs: 2
#    ORs: 1
#    XORs: 2
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(5)
OUTPUT(9)

4=XOR(1,2)
7=NOT(4)
5=XOR(4,3)
6=AND(3,4)
8=AND(2,7)
9=OR(6,8)
```

```
F0(x1,x2,x3) = (x1 + x2 + x3) * (x1 + x2' + x3') * (x1' + x2 + x3') * (x1' + x2' + x3)

F1(x1,x2,x3) = (x1 + x2 + x3) * (x1 + x2 + x3') * (x1 + x2' + x3) * (x1' + x2 + x3)
```

```
F0(x1,x2,x3) = x3 ^ x2 ^ x1

F1(x1,x2,x3) = x2x3 ^ x1x3 ^ x1x2
```

```
fileroot = adder1
Style: MAXTERM
Callback: [formuladnf]
```

24. BENCH ->Simulate

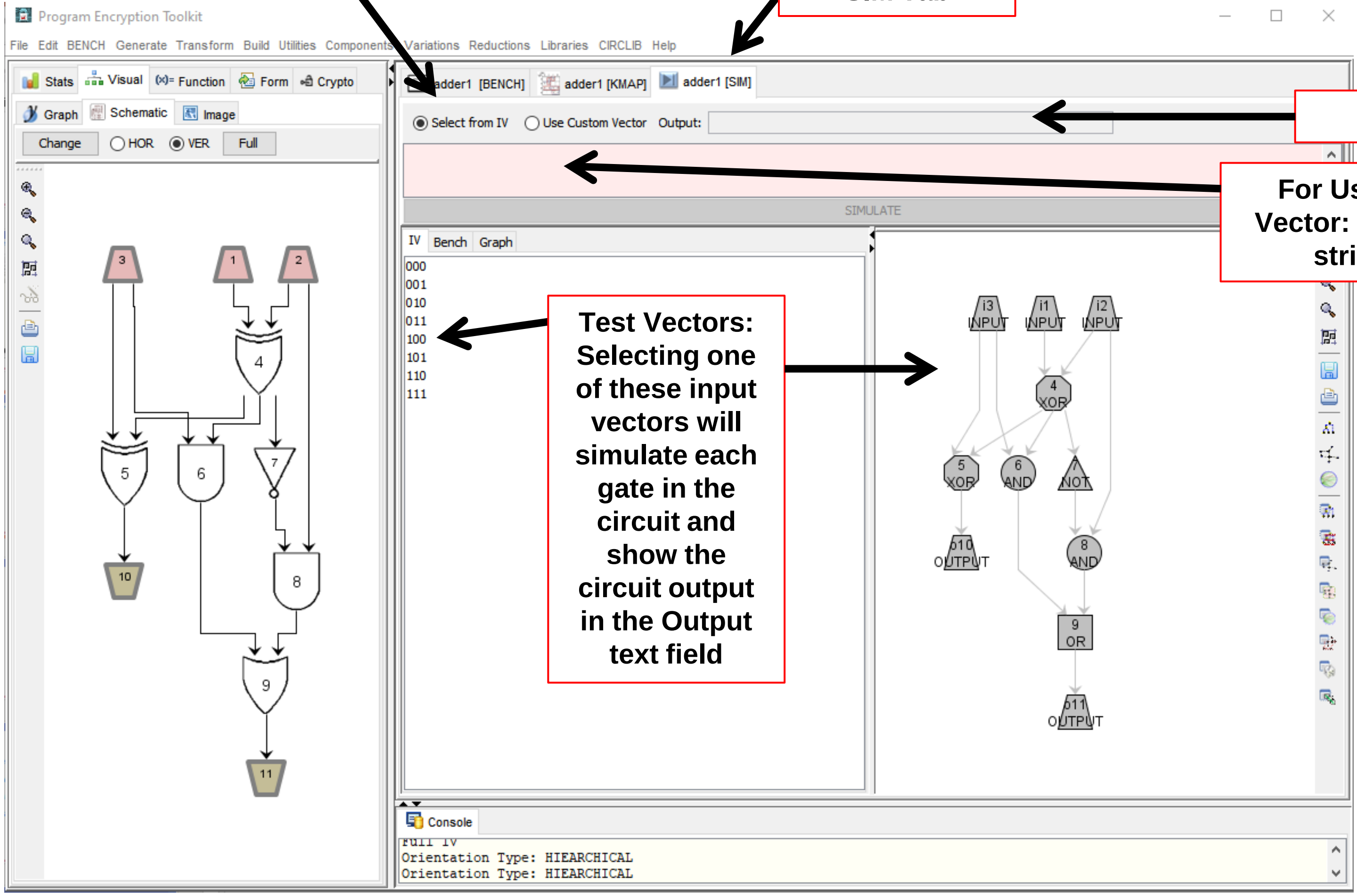


Use this for large input size circuits: it will generate random test vectors

To continue…

Use test vectors (IV)

SIM Tab


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto
Graph Schematic Image
Change HOR VER Full
adder1 [BENCH] adder1 [KMAP] adder1 [SIM]
Select from IV Use Custom Vector Output:
SIMULATE
IV Bench Graph
Console

```
000
001
010
011
100
101
110
111
```

```
Full IV
Orientation Type: HIEARCHICAL
Orientation Type: HIEARCHICAL
```

Circuit Output

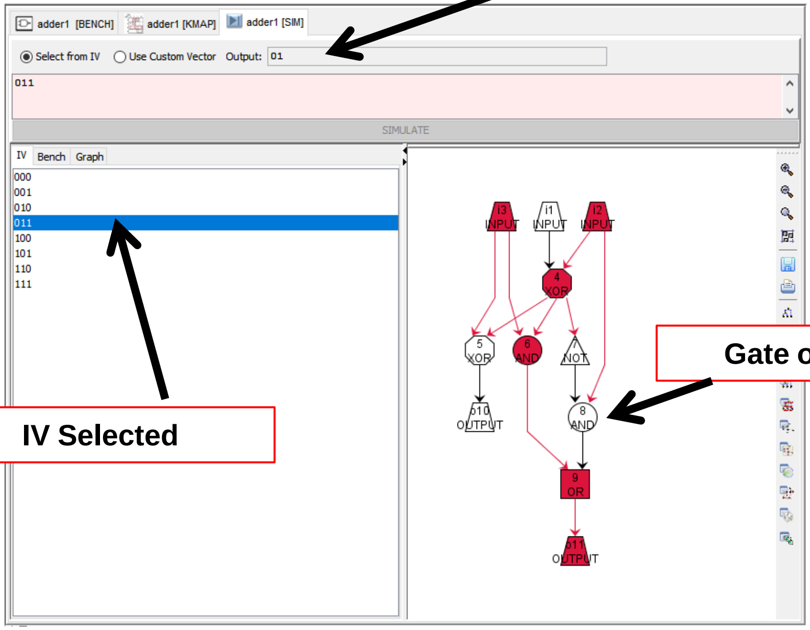
For Use Custom Vector: enter binary string here

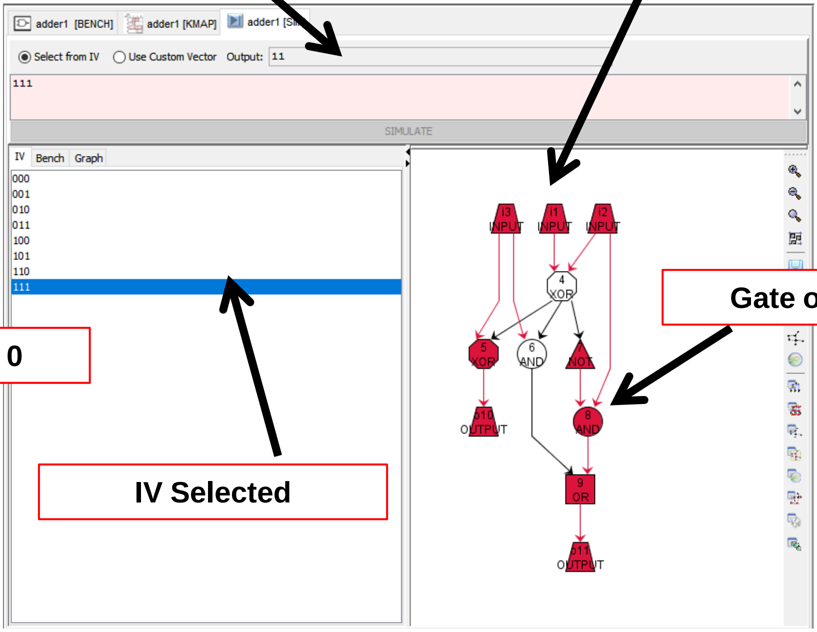
Test Vectors: Selecting one of these input vectors will simulate each gate in the circuit and show the circuit output in the Output text field

24. BENCH ->Simulate

Circult Output

INPUT = 111





Gate output = 0

Gate output =

IV Selected

IV Selected

Use Case 2: Create New BENCH File (Text)

1. **File -> New -> BENCH File**

   Browse to a path and provide a filename

2. Edit text in the text pane, entering a valid BENCH netlist

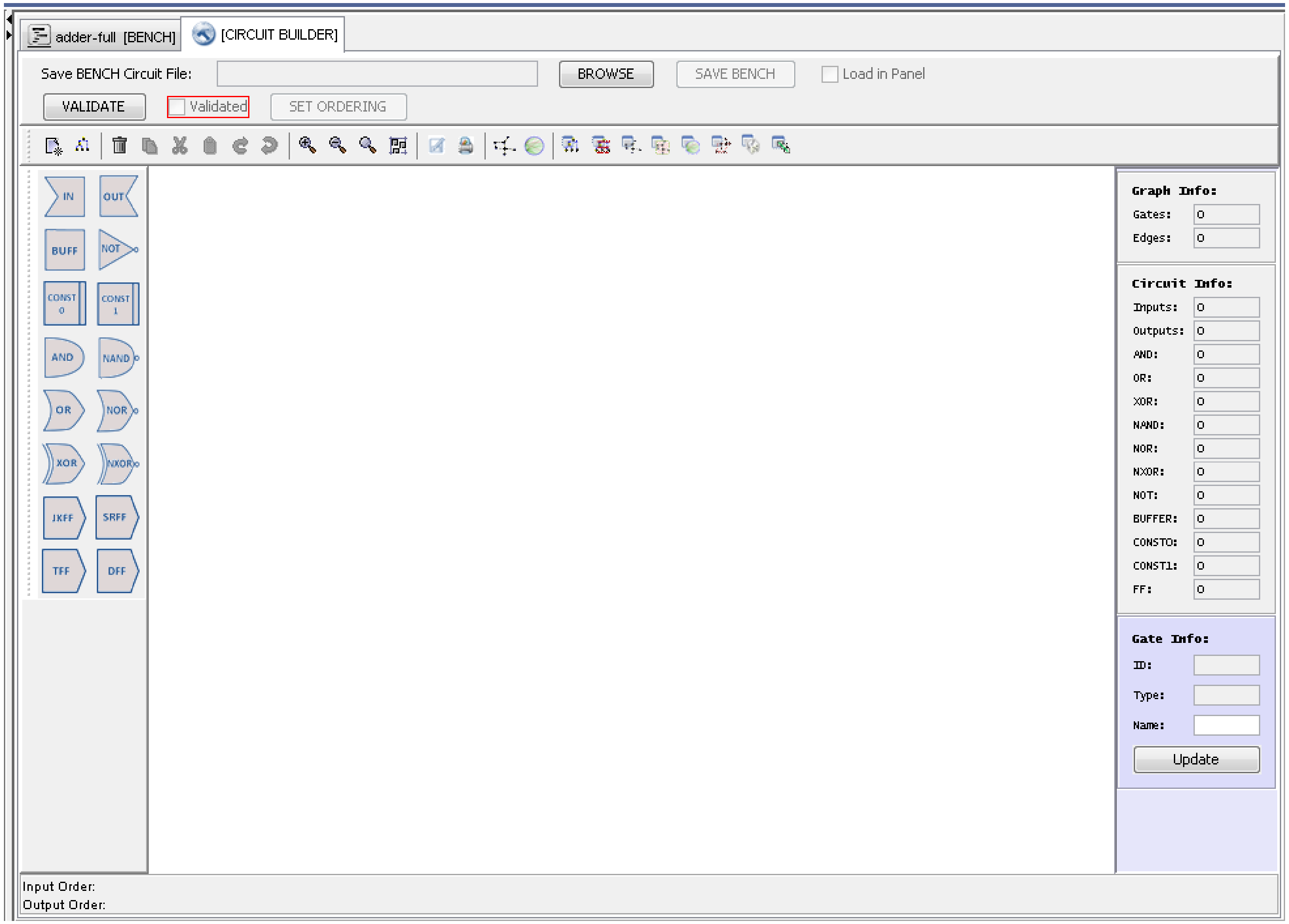
3. **File -> Save**, to save edits

4. **File -> Save As**, saves current contents to new file

5. **File -> Close**, closes the text edit panel

Use Case 3: Create New BENCH File (Visual)

- Build -> From Circuit Builder



Save BENCH Circuit File:

Load in Panel

Validated

Graph Info:

Gates:	0
Edges:	0

Circuit Info:

Inputs:	0
Outputs:	0
AND:	0
OR:	0
XOR:	0
NAND:	0
NOR:	0
NXOR:	0
NOT:	0
BUFFER:	0
CONST0:	0
CONST1:	0
FF:	0

Gate Info:

ID:

Type:

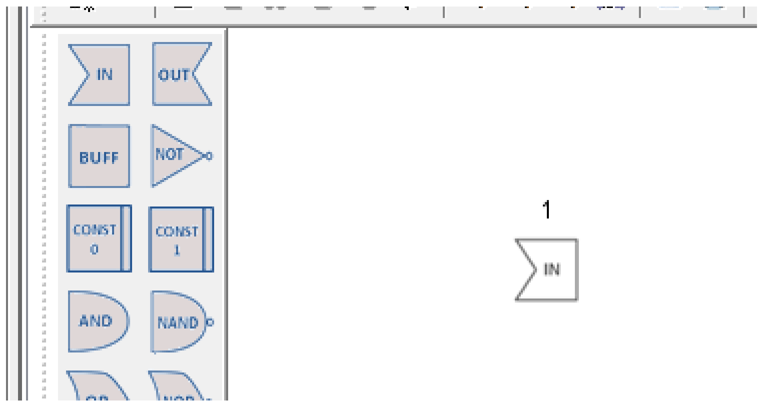
Name:

Input Order:

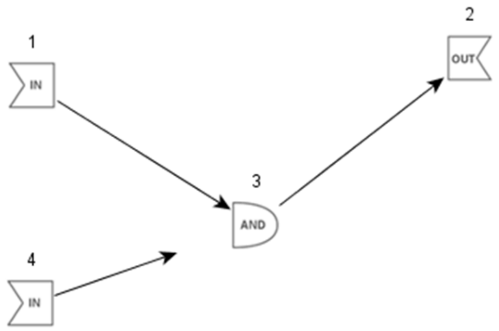
Output Order:

Use Case 3: Create New BENCH File (Visual)

1: Select gate type from palette (left click)



2: Left-click on canvas to drop a gate



3: Connect gates: left-click AND hold on a source gate, drag, then release on a target gate

4: Click “Validate”: errors are reported, otherwise Validated checkbox becomes selected

Use Case 3: Create New BENCH File (Visual)

- Left-click on gate = selects it, for moving/replacement
- Left-click on wire = selects it for adding bends
- Left-click on canvas = no current selection, adds a gate to the canvas
- Left-click on canvas = if a gate is selected, deselects any gate/wire

- Use cut/delete on selection to get rid of
- Use undo/rundo

NOTE: Copy/paste functionality does not work fully in Release 1.0

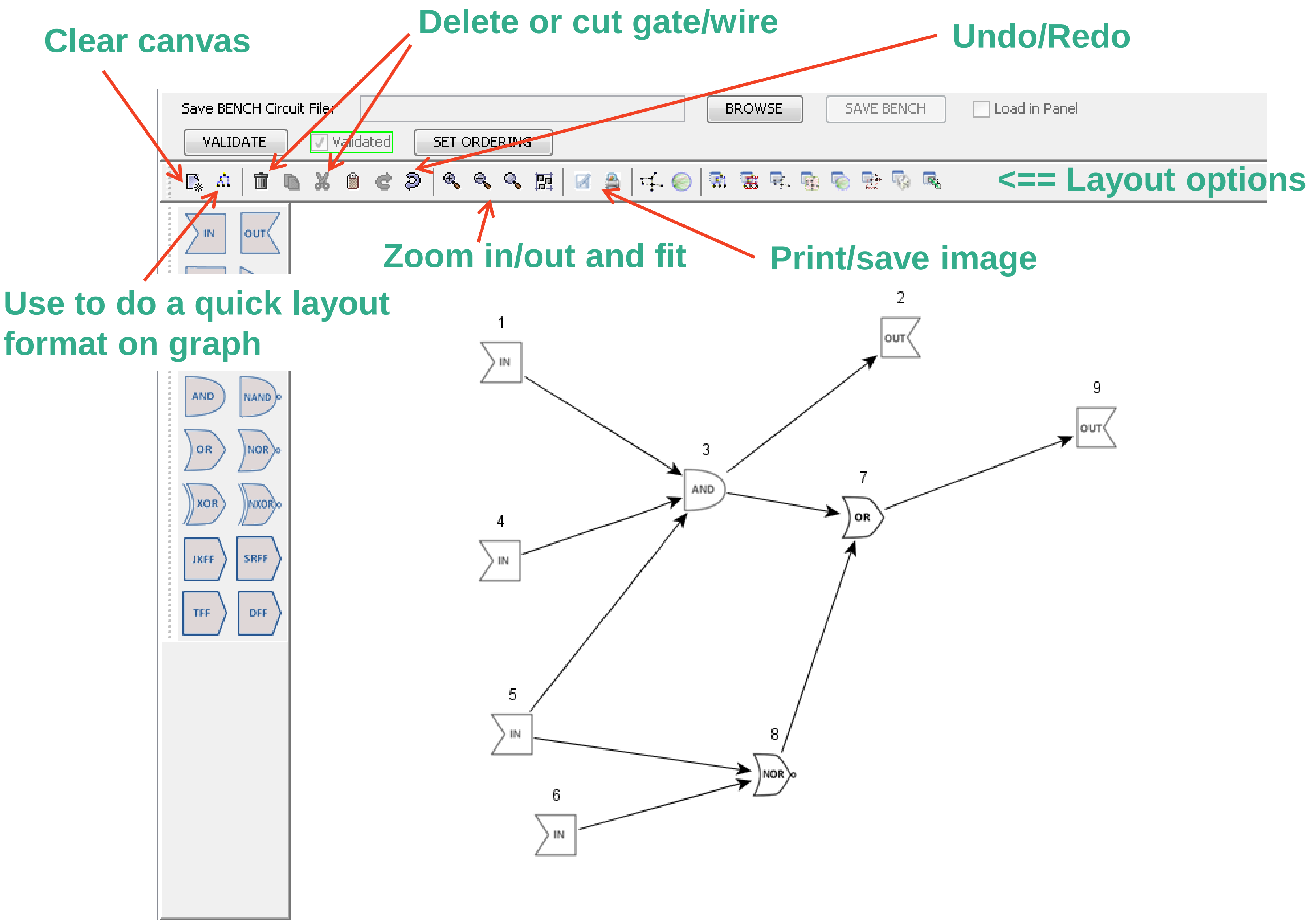
5: Click BROWSE to select a path and filename for the BENCH file to be saved

IF circuit is validated AND BENCH path has been chosen,
SAVE BENCH button is enabled and will write the file

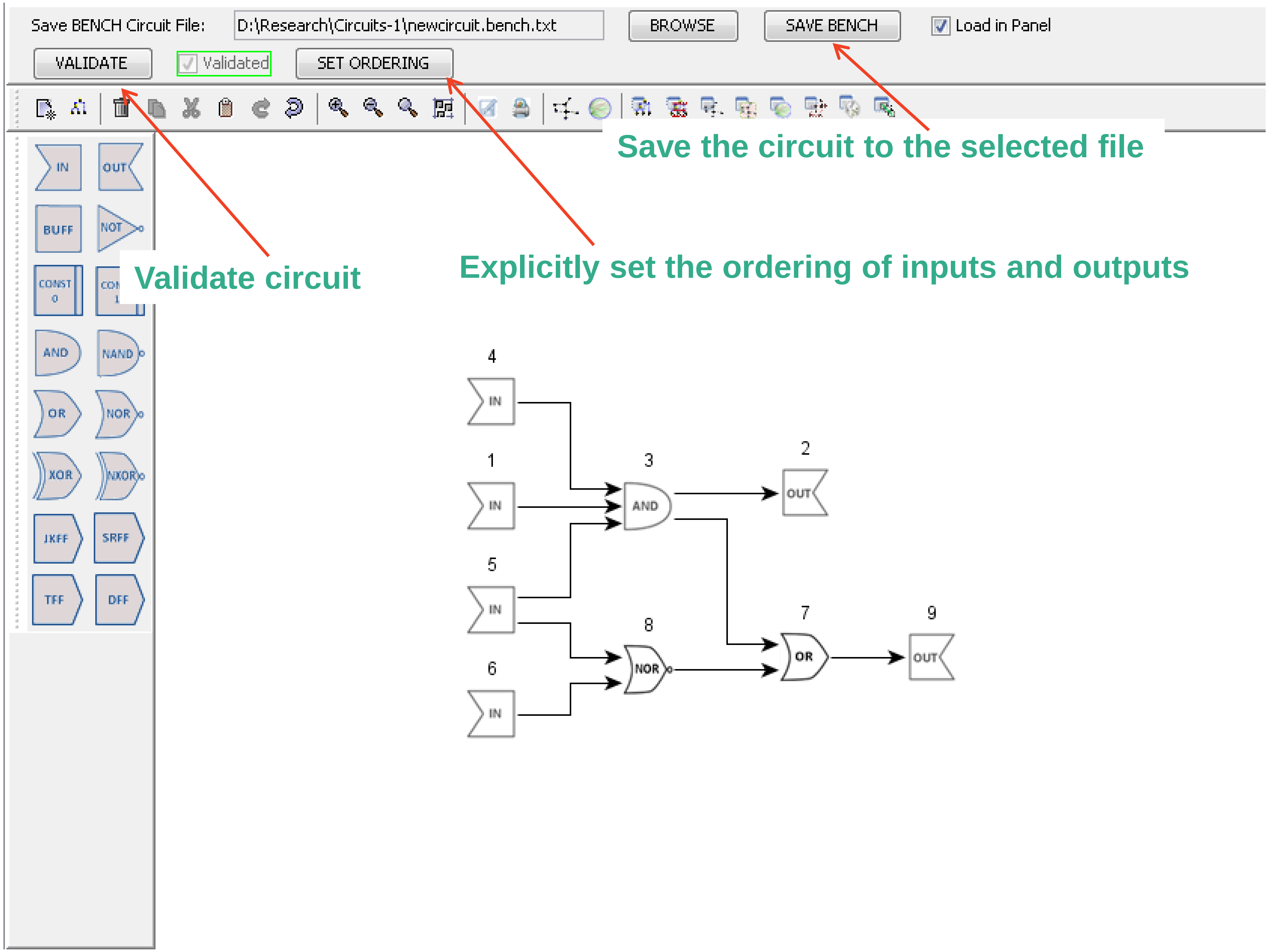
Check “Load in Panel” to also load the bench file into a text panel on save

Any changes to circuit will invalidate the circuit and you will need to revalidate it

Use Case 3: Create New BENCH File (Visual)



Use Case 3: Create New BENCH File (Visual)



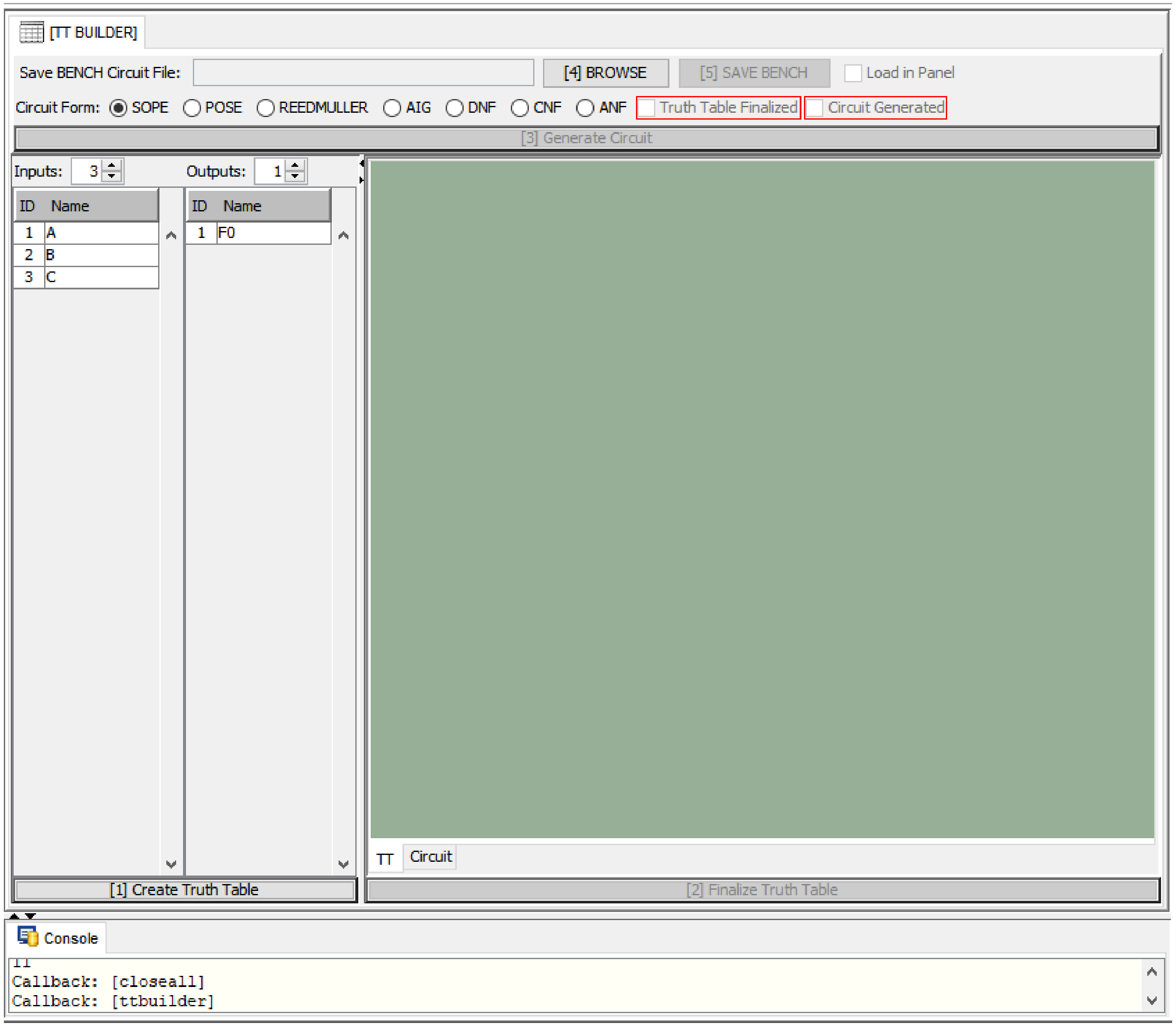
Save the circuit to the selected file

Validate circuit

Explicitly set the ordering of inputs and outputs

Use Case 4: Create New BENCH File (Truth Table)

- Build -> From Truth Table



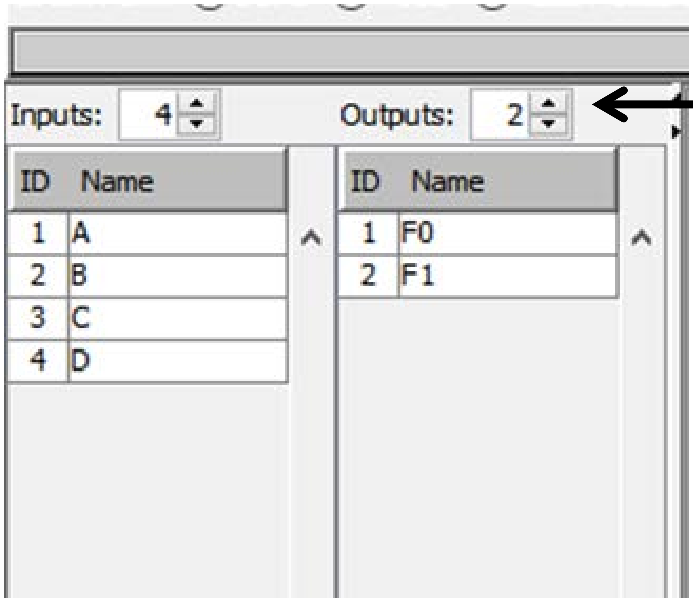
ID	Name
1	A
2	B
3	C

ID	Name
1	F0

```
TT
Callback: [closeall]
Callback: [ttbuilder]
```

Use Case 4: Create New BENCH File (Truth Table)

1: Select input and output size of truth table



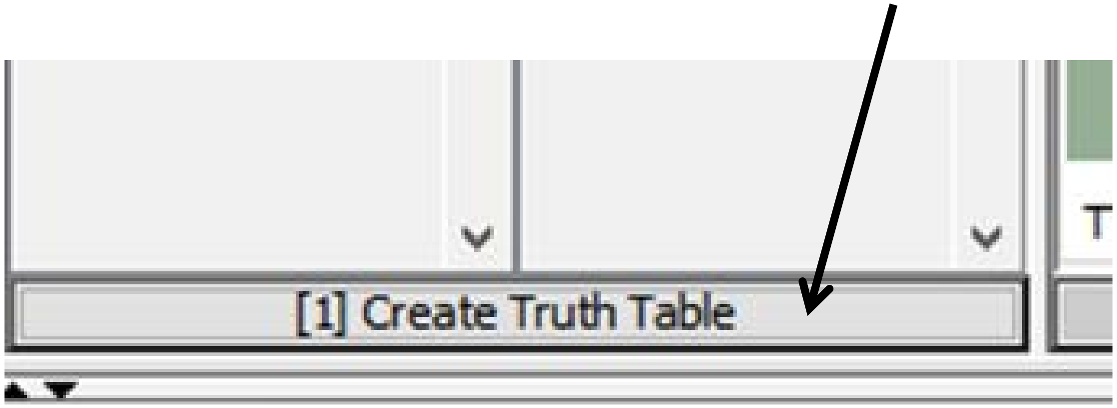
Inputs: 4 Outputs: 2

ID	Name
1	A
2	B
3	C
4	D

ID	Name
1	F0
2	F1

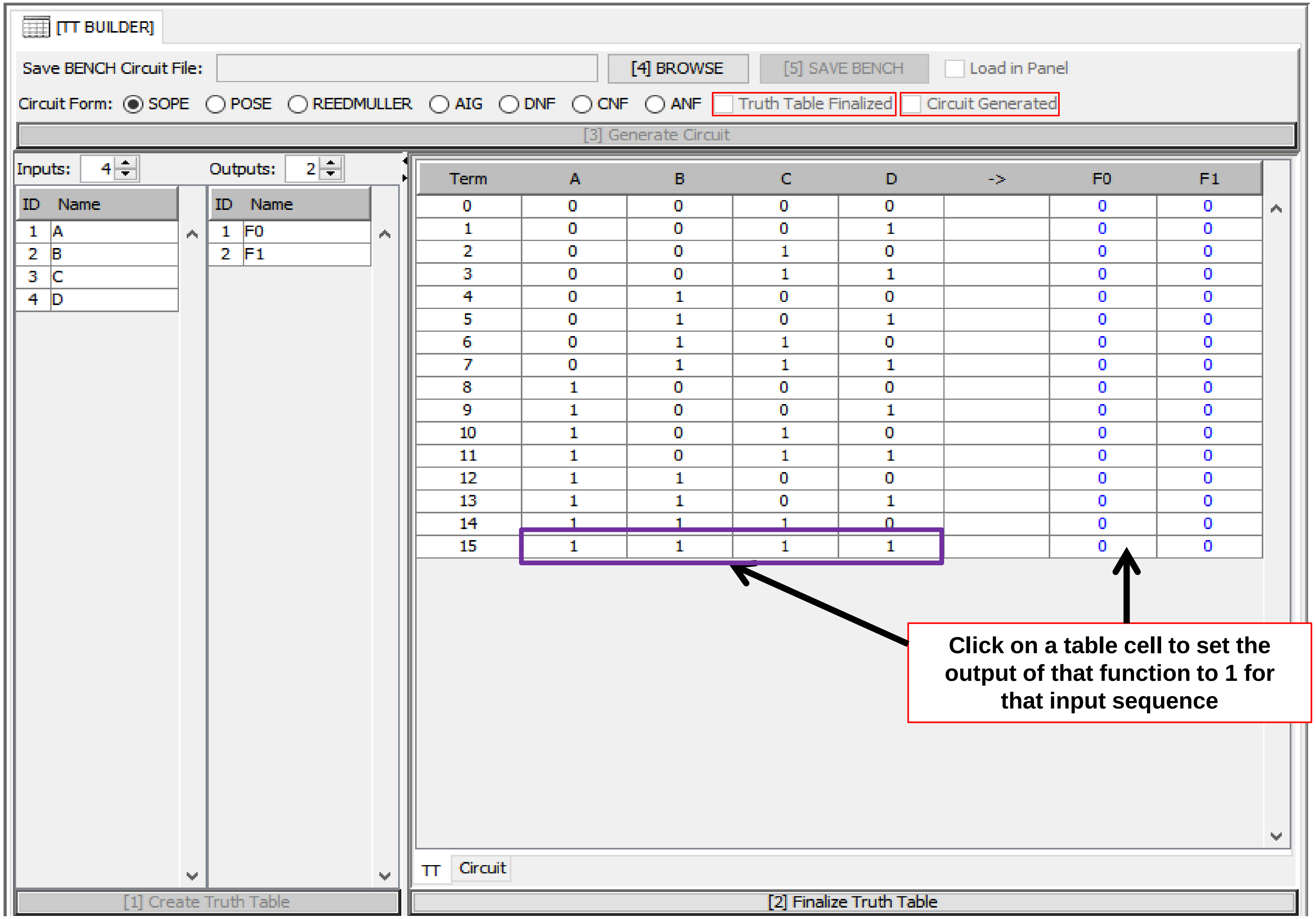
Select Number of Inputs/Outputs

2: Click Create Truth Table



Use Case 4: Create New BENCH File (Truth Table)

3: Specify Truth Table



Save BENCH Circuit File:

☐ Load in Panel

Circuit Form: [✓ SOPE] POSE REEDMULLER AIG DNF CNF ANF ☐ Truth Table Finalized ☐ Circuit Generated

Inputs: 4 Outputs: 2

ID	Name
1	A
2	B
3	C
4	D

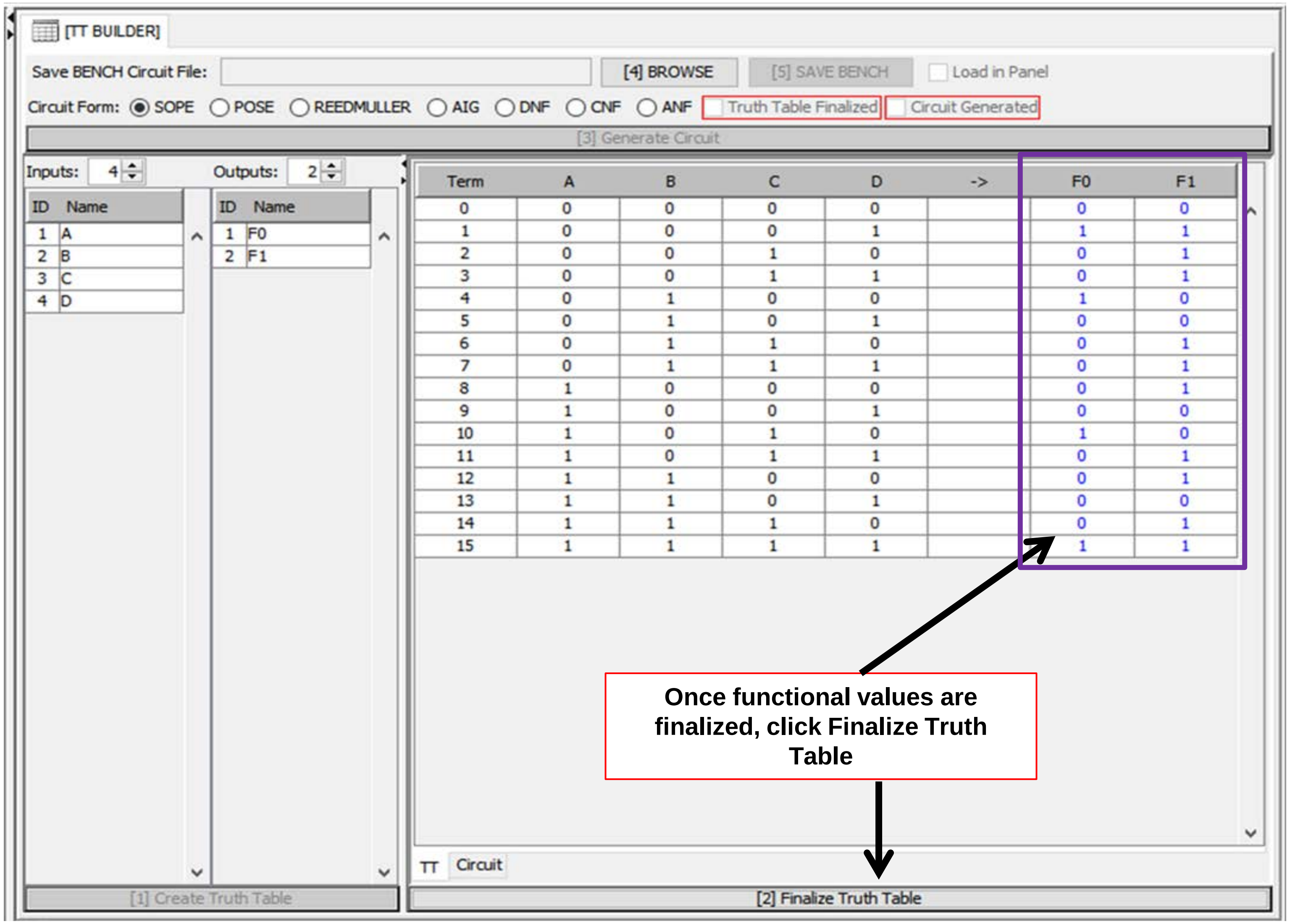
ID	Name
1	F0
2	F1

Term	A	B	C	D	->	F0	F1
0	0	0	0	0		0	0
1	0	0	0	1		0	0
2	0	0	1	0		0	0
3	0	0	1	1		0	0
4	0	1	0	0		0	0
5	0	1	0	1		0	0
6	0	1	1	0		0	0
7	0	1	1	1		0	0
8	1	0	0	0		0	0
9	1	0	0	1		0	0
10	1	0	1	0		0	0
11	1	0	1	1		0	0
12	1	1	0	0		0	0
13	1	1	0	1		0	0
14	1	1	1	0		0	0
15	1	1	1	1		0	0

Click on a table cell to set the output of that function to 1 for that input sequence

Use Case 4: Create New BENCH File (Truth Table)

4: Finalize Truth Table



Save BENCH Circuit File:

Load in Panel

Circuit Form: [✓ SOPE] POSE REEDMULLER AIG DNF CNF ANF

Truth Table Finalized

Circuit Generated

Inputs: 4

ID	Name
1	A
2	B
3	C
4	D

Outputs: 2

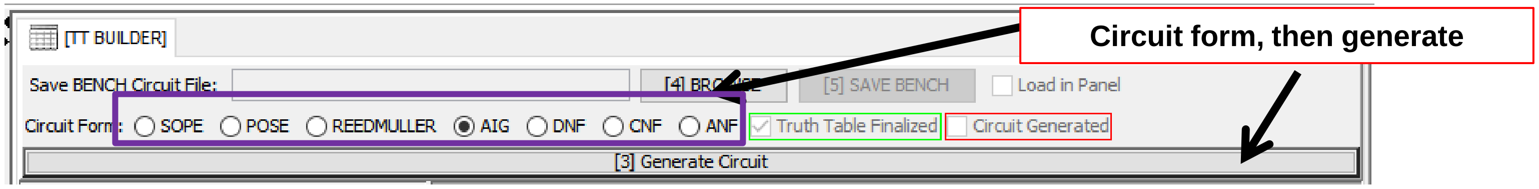
ID	Name
1	F0
2	F1

Term	A	B	C	D	->	F0	F1
0	0	0	0	0		0	0
1	0	0	0	1		1	1
2	0	0	1	0		0	1
3	0	0	1	1		0	1
4	0	1	0	0		1	0
5	0	1	0	1		0	0
6	0	1	1	0		0	1
7	0	1	1	1		0	1
8	1	0	0	0		0	1
9	1	0	0	1		0	0
10	1	0	1	0		1	0
11	1	0	1	1		0	1
12	1	1	0	0		0	1
13	1	1	0	1		0	0
14	1	1	1	0		0	1
15	1	1	1	1		1	1

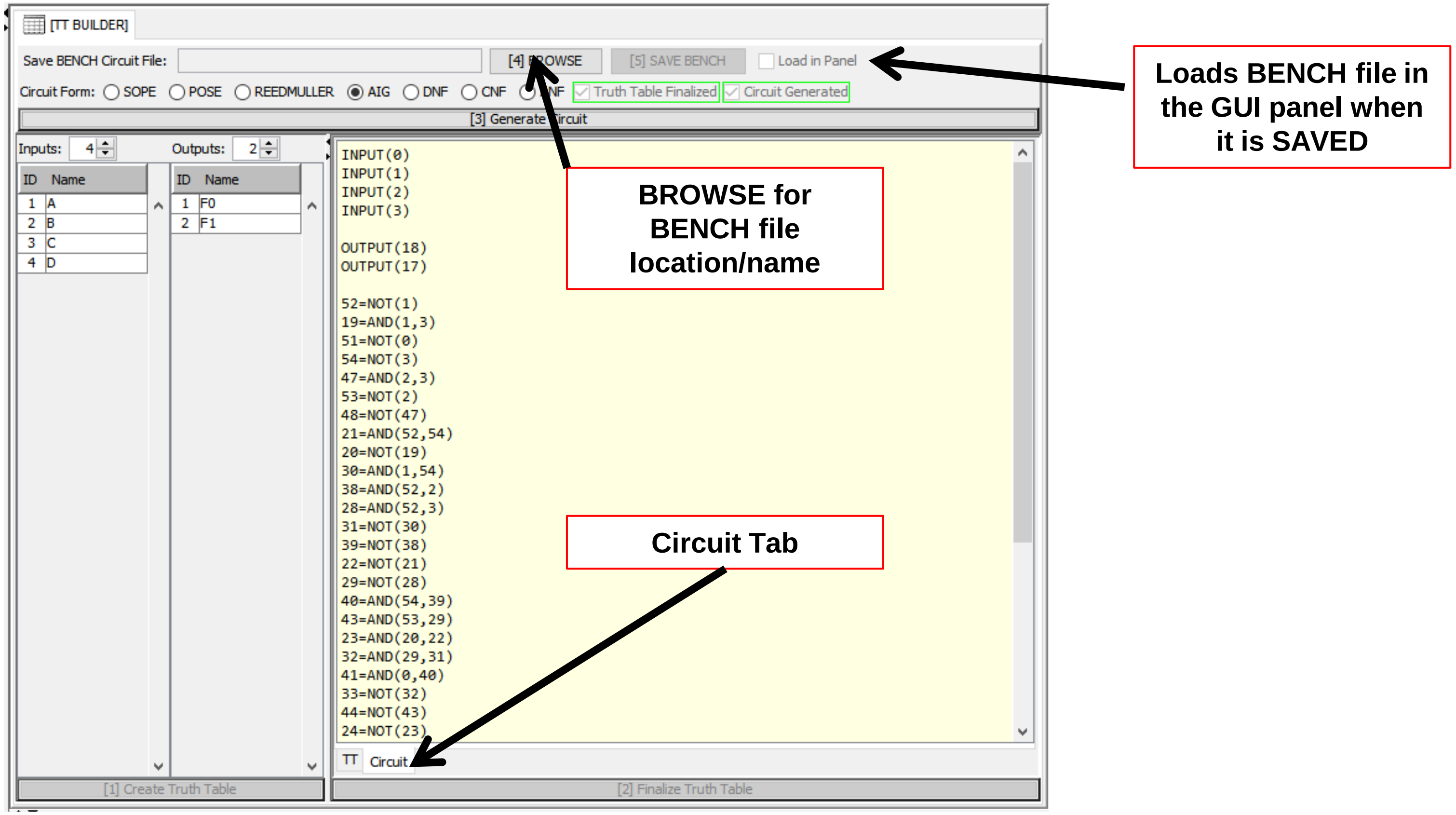
Once functional values are finalized, click Finalize Truth Table

Use Case 4: Create New BENCH File (Truth Table)

5: Pick which circuit form, then Generate Circuit



6: BENCH generated: browse for save file/select load in Panel



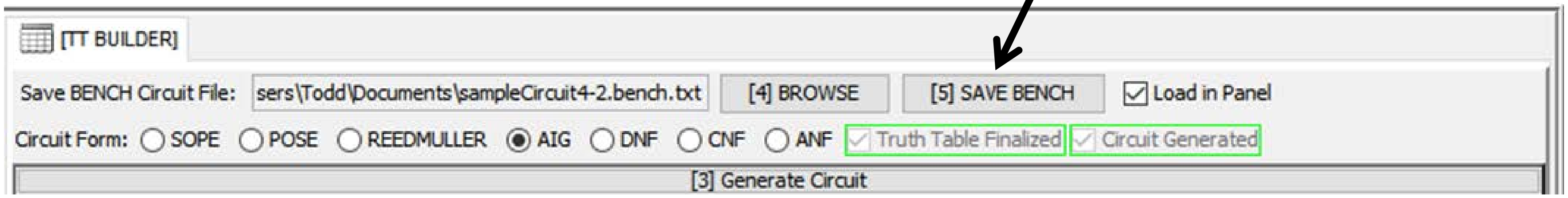
ID	Name
1	A
2	B
3	C
4	D

ID	Name
1	F0
2	F1

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(18)
OUTPUT(17)

52=NOT(1)
19=AND(1,3)
51=NOT(0)
54=NOT(3)
47=AND(2,3)
53=NOT(2)
48=NOT(47)
21=AND(52,54)
20=NOT(19)
30=AND(1,54)
38=AND(52,2)
28=AND(52,3)
31=NOT(30)
39=NOT(38)
22=NOT(21)
29=NOT(28)
40=AND(54,39)
43=AND(53,29)
23=AND(20,22)
32=AND(29,31)
41=AND(0,40)
33=NOT(32)
44=NOT(43)
24=NOT(23)
```

Use Case 4: Create New BENCH File (Truth Table)

7: Once BENCH file is selected, click SAVE BENCH

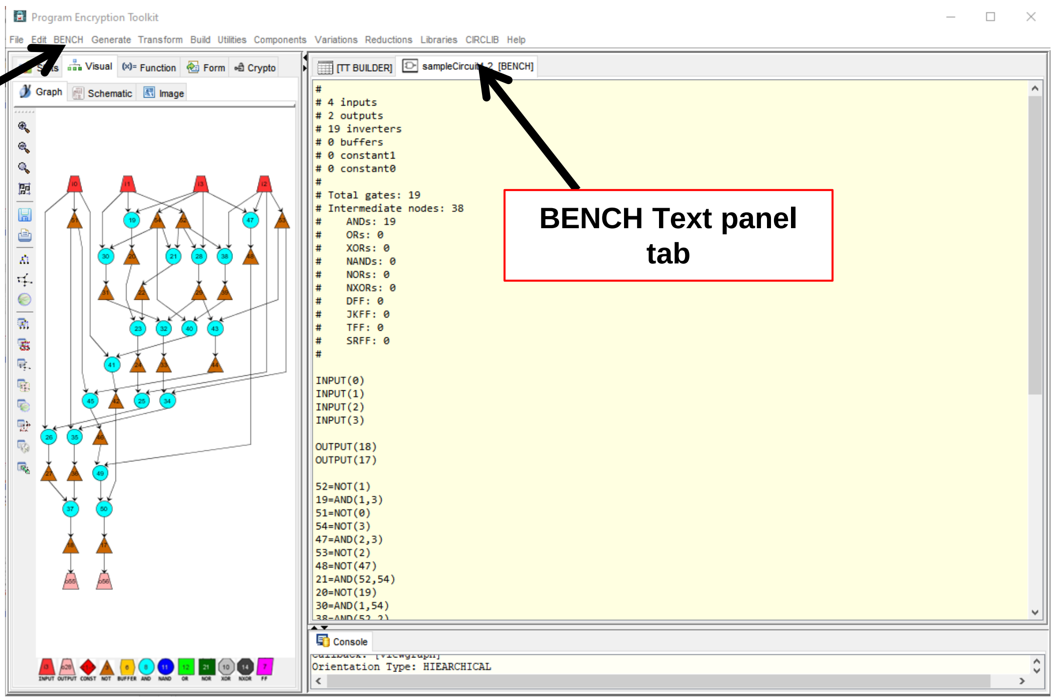


Save BENCH Circuit File: sers\Todd\Documents\sampleCircuit4-2.bench.txt

[✓ Load in Panel]

Circuit Form: SOPE POSE REEDMULLER [● AIG] DNF CNF ANF [✓ Truth Table Finalized] [✓ Circuit Generated]

8: If Load in Panel selected, BENCH text panel for file will appear as well


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Visual
Function
Form
Crypto
Graph
Schematic
Image
[TT BUILDER]
sampleCircuit4-2 [BENCH]
Console
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF

```
#
# 4 inputs
# 2 outputs
# 19 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 19
# Intermediate nodes: 38
#    ANDs: 19
#    ORs: 0
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(18)
OUTPUT(17)

52=NOT(1)
19=AND(1,3)
51=NOT(0)
54=NOT(3)
47=AND(2,3)
53=NOT(2)
48=NOT(47)
21=AND(52,54)
20=NOT(19)
30=AND(1,54)
38=AND(52,2)
```

```
Callback: [ViewGraph]
Orientation Type: HIEARCHICAL
```

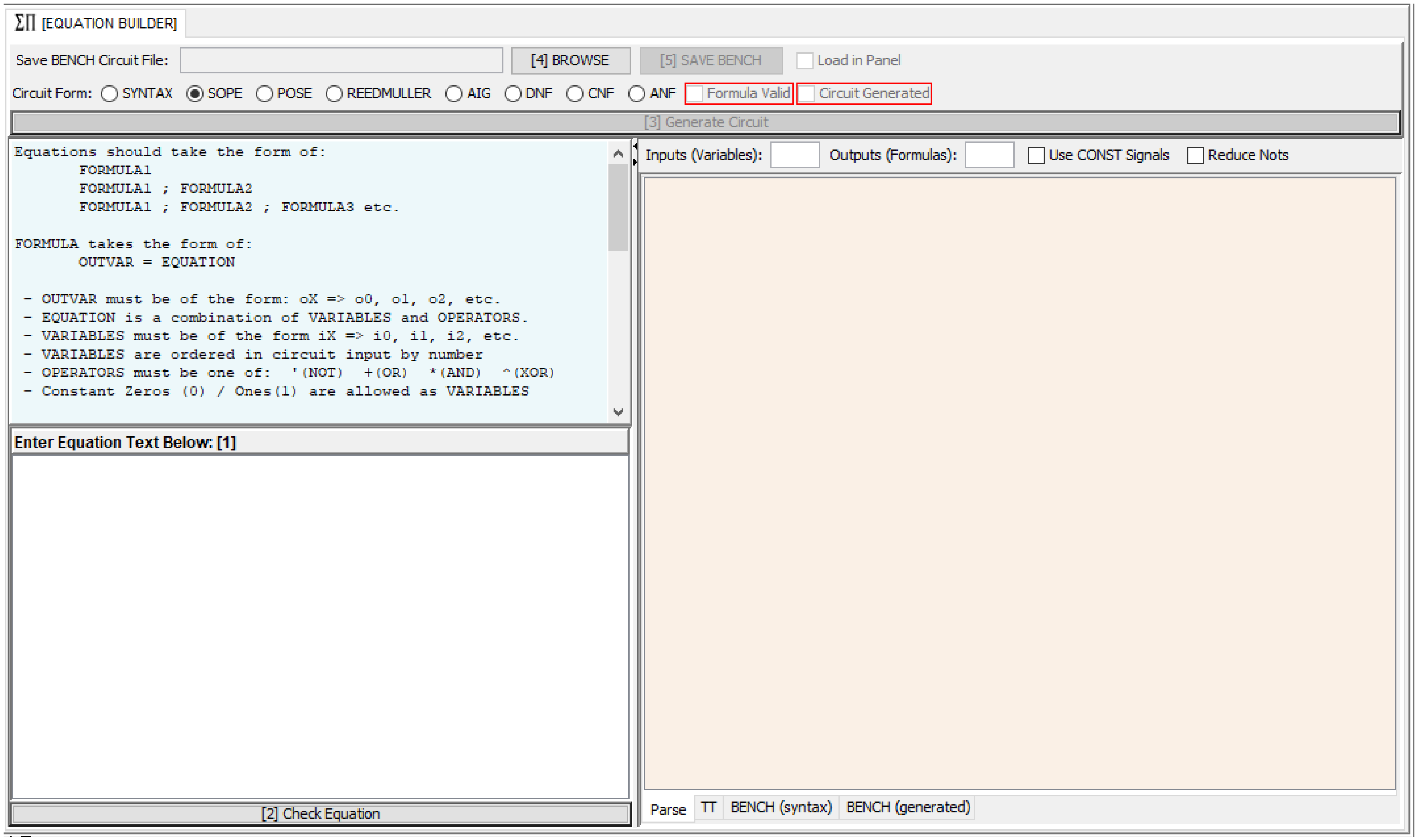
BENCH Text panel tab

Still much do BENCH->Compile Combinational

Then BENCH->View Graph to see created circuit

Use Case 5: Create New BENCH File (Equation)

- Build -> From Equation



ΣΠ [EQUATION BUILDER]

Save BENCH Circuit File: [4] BROWSE [5] SAVE BENCH ☐ Load in Panel

Circuit Form: ○ SYNTAX [✓ SOPE] ○ POSE ○ REEDMULLER ○ AIG ○ DNF ○ CNF ○ ANF ☐ Formula Valid ☐ Circuit Generated

[3] Generate Circuit

```
Equations should take the form of:
       FORMULA1
       FORMULA1 ; FORMULA2
       FORMULA1 ; FORMULA2 ; FORMULA3 etc.

FORMULA takes the form of:
       OUTVAR = EQUATION

 - OUTVAR must be of the form: oX => o0, o1, o2, etc.
 - EQUATION is a combination of VARIABLES and OPERATORS.
 - VARIABLES must be of the form iX => i0, i1, i2, etc.
 - VARIABLES are ordered in circuit input by number
 - OPERATORS must be one of:  '(NOT)  +(OR)  *(AND)  ^(XOR)
 - Constant Zeros (0) / Ones(1) are allowed as VARIABLES
```

Enter Equation Text Below: [1]

[2] Check Equation

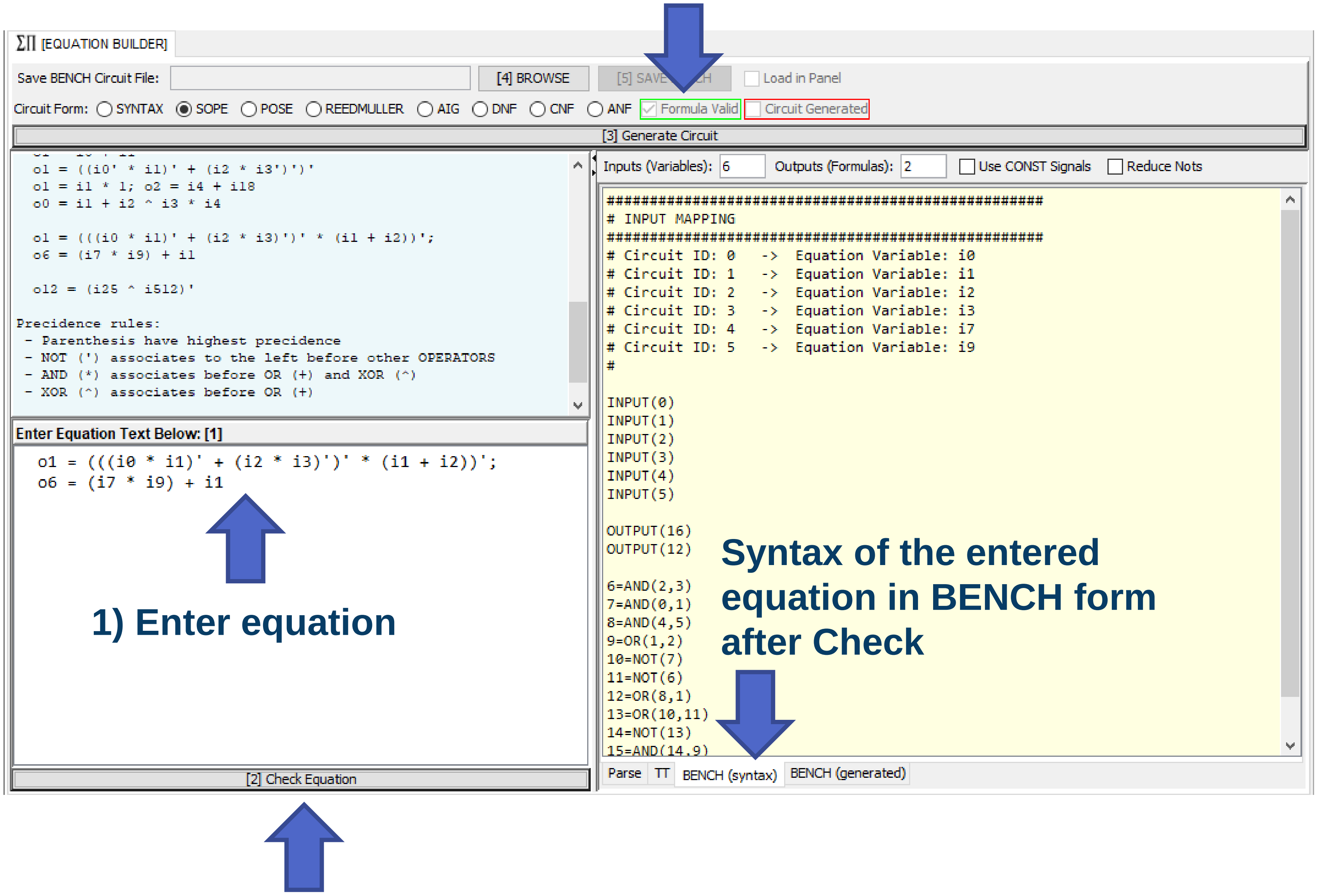
Inputs (Variables): Outputs (Formulas): ☐ Use CONST Signals ☐ Reduce Nots

Use Case 5: Create New BENCH File (Equation)

```
Equations should take the form of:
        FORMULA1
        FORMULA1 ; FORMULA2
        FORMULA1 ; FORMULA2 ; FORMULA3 etc.

FORMULA takes the form of:
        OUTVAR = EQUATION

 - OUTVAR must be of the form: oX => o0, o1, o2, etc.
 - EQUATION is a combination of VARIABLES and OPERATORS.
 - VARIABLES must be of the form iX => i0, i1, i2, etc.
 - VARIABLES are ordered in circuit input by number
 - OPERATORS must be one of:  '(NOT)  +(OR)  *(AND)  ^(XOR)
 - Constant Zeros (0) / Ones(1) are allowed as VARIABLES

General rules:
 - Formulas must be separated by a semicolon if more than 1
 - The last (or if there is only 1) formula does not need a ;
 - Formulas can be on separate lines (separated by a NL)
 - At least 1 VARIABLE required (o1 = 0/o1 = 1 not allowed)
 - Use parenthesis to clarify logical expressions and precedence


Examples:
  o1 = i0 + i1
  o1 = ((i0' * i1)' + (i2 * i3')')'
  o1 = i1 * 1; o2 = i4 + i18
  o0 = i1 + i2 ^ i3 * i4

  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1

  o12 = (i25 ^ i512)'

Precedence rules:
 - Parenthesis have highest precedence
 - NOT (') associates to the left before other OPERATORS
 - AND (*) associates before OR (+) and XOR (^)
 - XOR (^) associates before OR (+)

Example:              o0 = i1 + i2' ^ i3 * i4
    is equivalent to: o0 = (i1 + ((i2') ^ (i3 * i4))
```

Use Case 5: Create New BENCH File (Equation)

For valid formula



Save BENCH Circuit File:

Circuit Form: ○ SYNTAX [✓ SOPE] ○ POSE ○ REEDMULLER ○ AIG ○ DNF ○ CNF ○ ANF ✓ Formula Valid ☐ Circuit Generated

```
  o1 = ((i0' * i1)' + (i2 * i3')')'
  o1 = i1 * 1; o2 = i4 + i18
  o0 = i1 + i2 ^ i3 * i4

  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1

  o12 = (i25 ^ i512)'

Precidence rules:
 - Parenthesis have highest precidence
 - NOT (') associates to the left before other OPERATORS
 - AND (*) associates before OR (+) and XOR (^)
 - XOR (^) associates before OR (+)
```

Enter Equation Text Below: [1]

```
  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1
```

Inputs (Variables): 6 Outputs (Formulas): 2 ☐ Use CONST Signals ☐ Reduce Nots

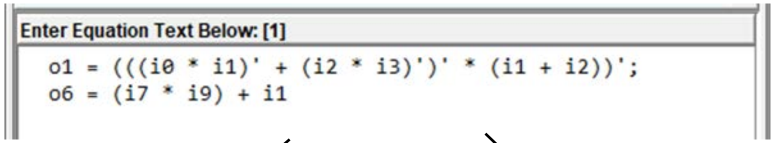
```
#########################################################
# INPUT MAPPING
#########################################################
# Circuit ID: 0   ->  Equation Variable: i0
# Circuit ID: 1   ->  Equation Variable: i1
# Circuit ID: 2   ->  Equation Variable: i2
# Circuit ID: 3   ->  Equation Variable: i3
# Circuit ID: 4   ->  Equation Variable: i7
# Circuit ID: 5   ->  Equation Variable: i9
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(12)

6=AND(2,3)
7=AND(0,1)
8=AND(4,5)
9=OR(1,2)
10=NOT(7)
11=NOT(6)
12=OR(8,1)
13=OR(10,11)
14=NOT(13)
15=AND(14,9)
```

1) Enter equation

Syntax of the entered equation in BENCH form after Check

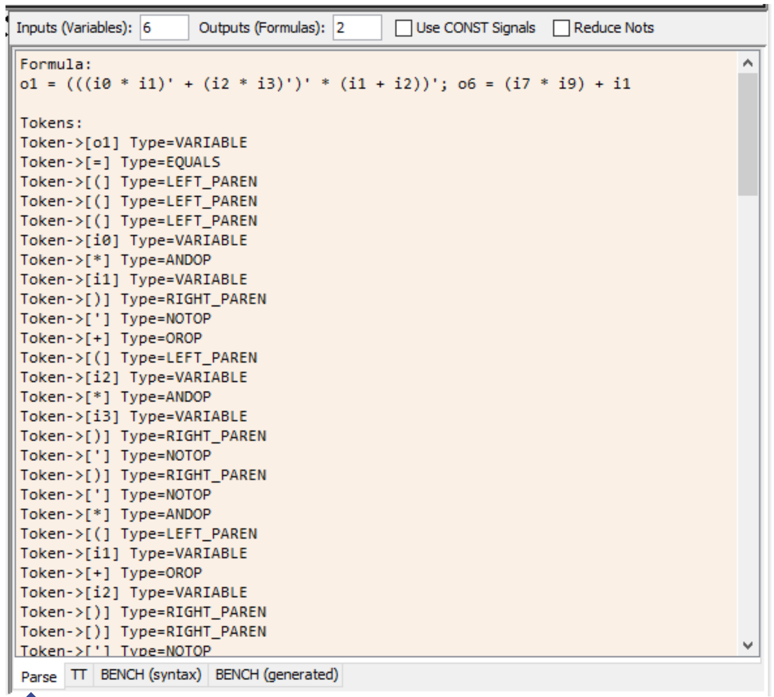
2) Check

Use Case 5: Create New BENCH File (Equation)



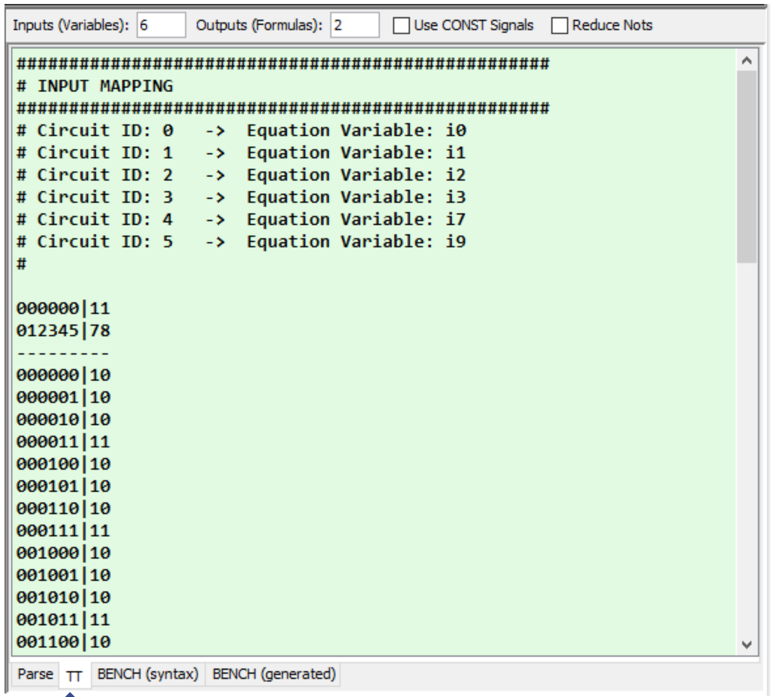
Enter Equation Text Below: [1]

```
o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
o6 = (i7 * i9) + i1
```



Inputs (Variables): 6 Outputs (Formulas): 2 ☐ Use CONST Signals ☐ Reduce Nots

```
Formula:
o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))'; o6 = (i7 * i9) + i1

Tokens:
Token->[o1] Type=VARIABLE
Token->[=] Type=EQUALS
Token->[(] Type=LEFT_PAREN
Token->[(] Type=LEFT_PAREN
Token->[(] Type=LEFT_PAREN
Token->[i0] Type=VARIABLE
Token->[*] Type=ANDOP
Token->[i1] Type=VARIABLE
Token->[)] Type=RIGHT_PAREN
Token->['] Type=NOTOP
Token->[+] Type=OROP
Token->[(] Type=LEFT_PAREN
Token->[i2] Type=VARIABLE
Token->[*] Type=ANDOP
Token->[i3] Type=VARIABLE
Token->[)] Type=RIGHT_PAREN
Token->['] Type=NOTOP
Token->[)] Type=RIGHT_PAREN
Token->['] Type=NOTOP
Token->[*] Type=ANDOP
Token->[(] Type=LEFT_PAREN
Token->[i1] Type=VARIABLE
Token->[+] Type=OROP
Token->[i2] Type=VARIABLE
Token->[)] Type=RIGHT_PAREN
Token->[)] Type=RIGHT_PAREN
Token->['] Type=NOTOP
```



Inputs (Variables): 6 Outputs (Formulas): 2 ☐ Use CONST Signals ☐ Reduce Nots

```
##################################################
# INPUT MAPPING
##################################################
# Circuit ID: 0   ->  Equation Variable: i0
# Circuit ID: 1   ->  Equation Variable: i1
# Circuit ID: 2   ->  Equation Variable: i2
# Circuit ID: 3   ->  Equation Variable: i3
# Circuit ID: 4   ->  Equation Variable: i7
# Circuit ID: 5   ->  Equation Variable: i9
#

000000|11
012345|78
---------
000000|10
000001|10
000010|10
000011|11
000100|10
000101|10
000110|10
000111|11
001000|10
001001|10
001010|10
001011|11
001100|10
```

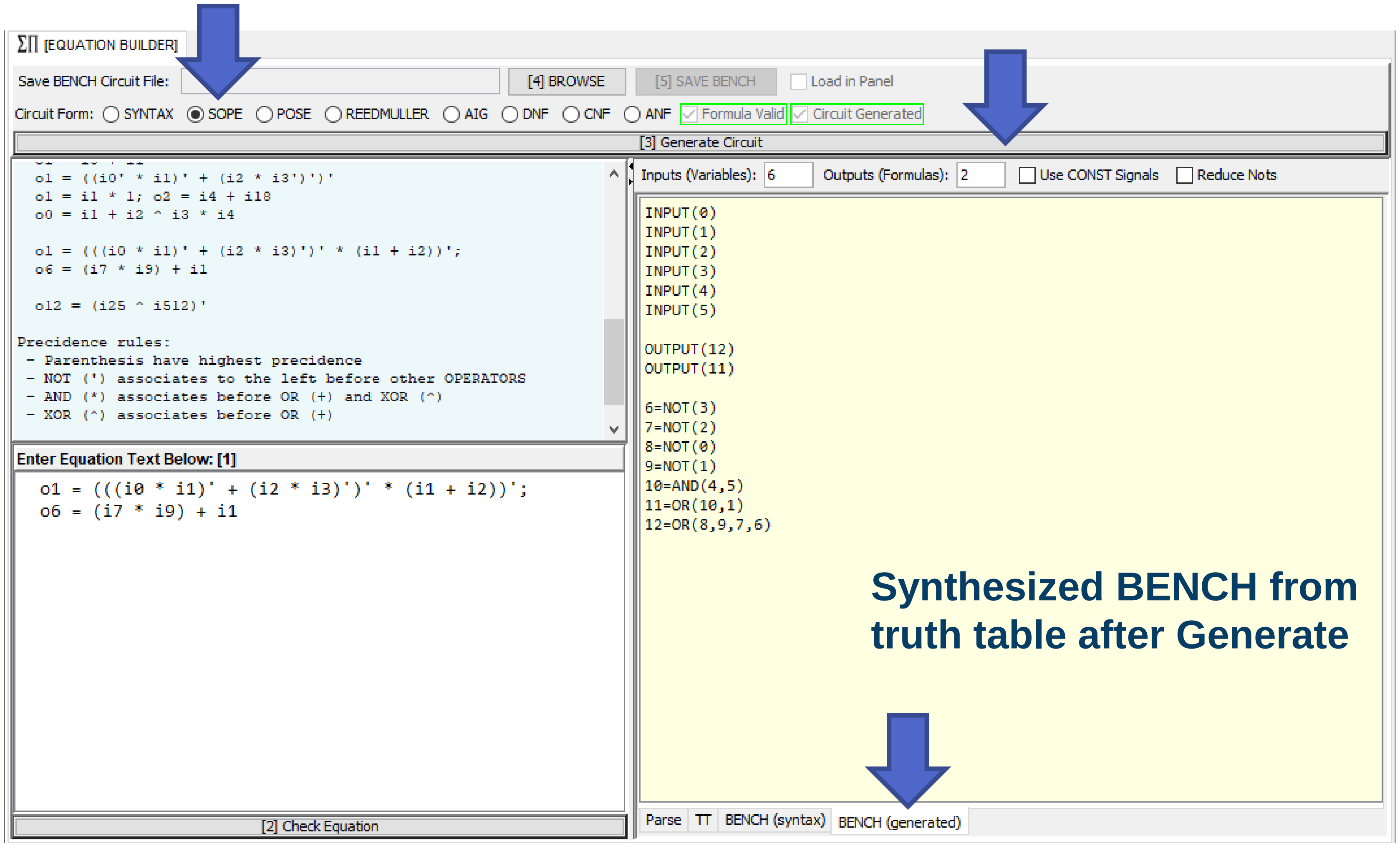
Parse of the entered Boolean formula

Truth table of entered Boolean formula

Use Case 5: Create New BENCH File (Equation)

3) Choose synthesized circuit form

4) Click Generate Circuit



Save BENCH Circuit File:

Load in Panel

Circuit Form: SYNTAX [✓ SOPE] POSE REEDMULLER AIG DNF CNF ANF ✓ Formula Valid ✓ Circuit Generated

```
  o1 = ((i0' * i1)' + (i2 * i3')')'
  o1 = i1 * 1; o2 = i4 + i18
  o0 = i1 + i2 ^ i3 * i4

  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1

  o12 = (i25 ^ i512)'

Precidence rules:
 - Parenthesis have highest precidence
 - NOT (') associates to the left before other OPERATORS
 - AND (*) associates before OR (+) and XOR (^)
 - XOR (^) associates before OR (+)
```

Enter Equation Text Below: [1]

```
  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1
```

Inputs (Variables): 6 Outputs (Formulas): 2 Use CONST Signals Reduce Nots

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(12)
OUTPUT(11)

6=NOT(3)
7=NOT(2)
8=NOT(0)
9=NOT(1)
10=AND(4,5)
11=OR(10,1)
12=OR(8,9,7,6)
```

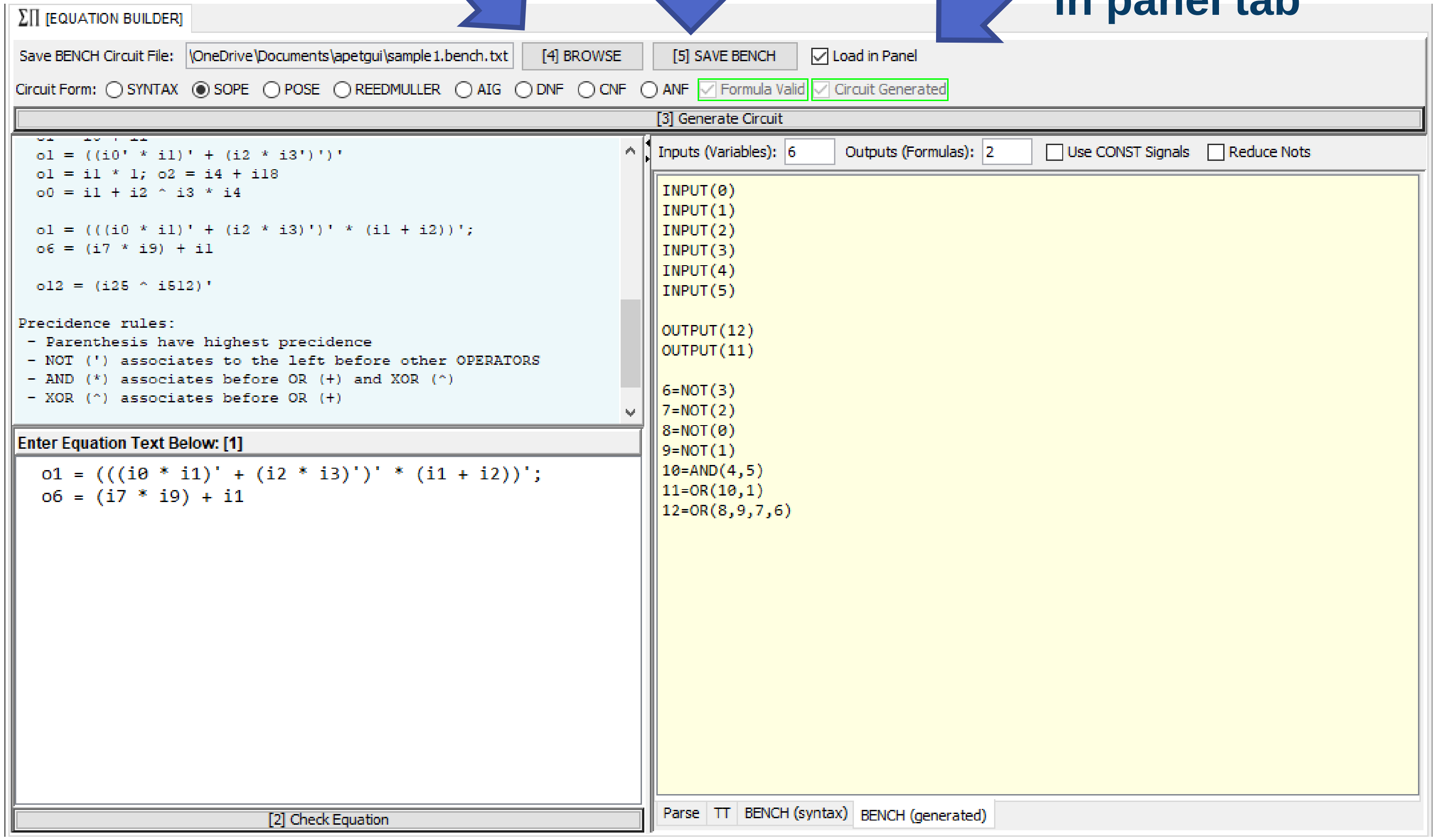
Synthesized BENCH from truth table after Generate

Use Case 5: Create New BENCH File (Equation)

5) BROWSE and pick filepath for BENCH file

7) Click SAVE BENCH to write out BENCH

6) Click Load in Panel to load saved BENCH in panel tab



Save BENCH Circuit File: \OneDrive\Documents\apetgui\sample1.bench.txt

☑ Load in Panel

Circuit Form: ○ SYNTAX [● SOPE] ○ POSE ○ REEDMULLER ○ AIG ○ DNF ○ CNF ○ ANF ☑ Formula Valid ☑ Circuit Generated

```
  o1 = ((i0' * i1)' + (i2 * i3')')'
  o1 = i1 * 1; o2 = i4 + i18
  o0 = i1 + i2 ^ i3 * i4

  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1

  o12 = (i25 ^ i512)'

Precidence rules:
 - Parenthesis have highest precidence
 - NOT (') associates to the left before other OPERATORS
 - AND (*) associates before OR (+) and XOR (^)
 - XOR (^) associates before OR (+)
```

Enter Equation Text Below: [1]

```
  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))';
  o6 = (i7 * i9) + i1
```

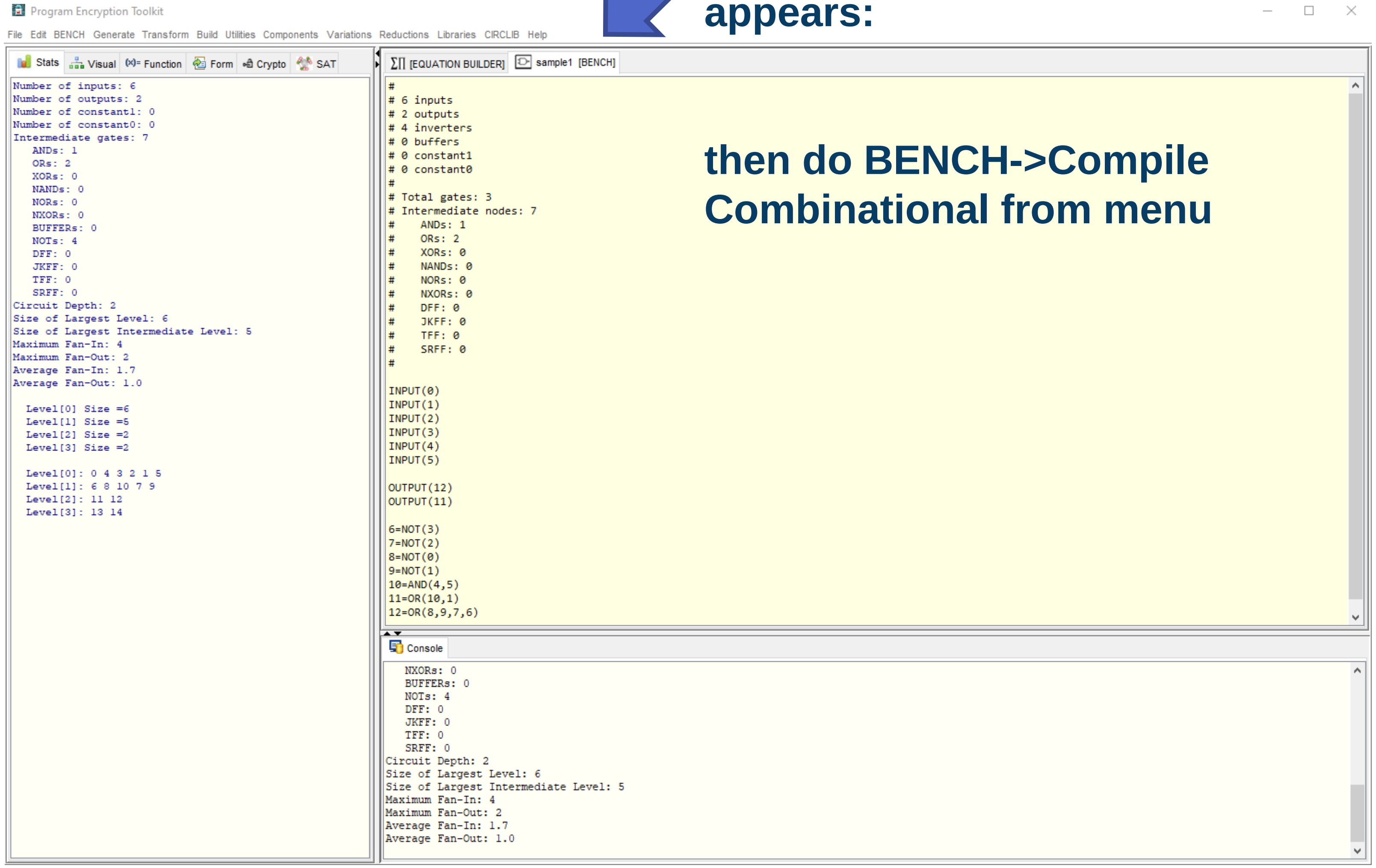
Inputs (Variables): 6 Outputs (Formulas): 2 ☐ Use CONST Signals ☐ Reduce Nots

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(12)
OUTPUT(11)

6=NOT(3)
7=NOT(2)
8=NOT(0)
9=NOT(1)
10=AND(4,5)
11=OR(10,1)
12=OR(8,9,7,6)
```

Use Case 5: Create New BENCH File (Equation)

8) After SAVE BENCH and load in panel, new BENCH tab appears:

then do BENCH->Compile Combinational from menu


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
[EQUATION BUILDER] sample1 [BENCH]
Console

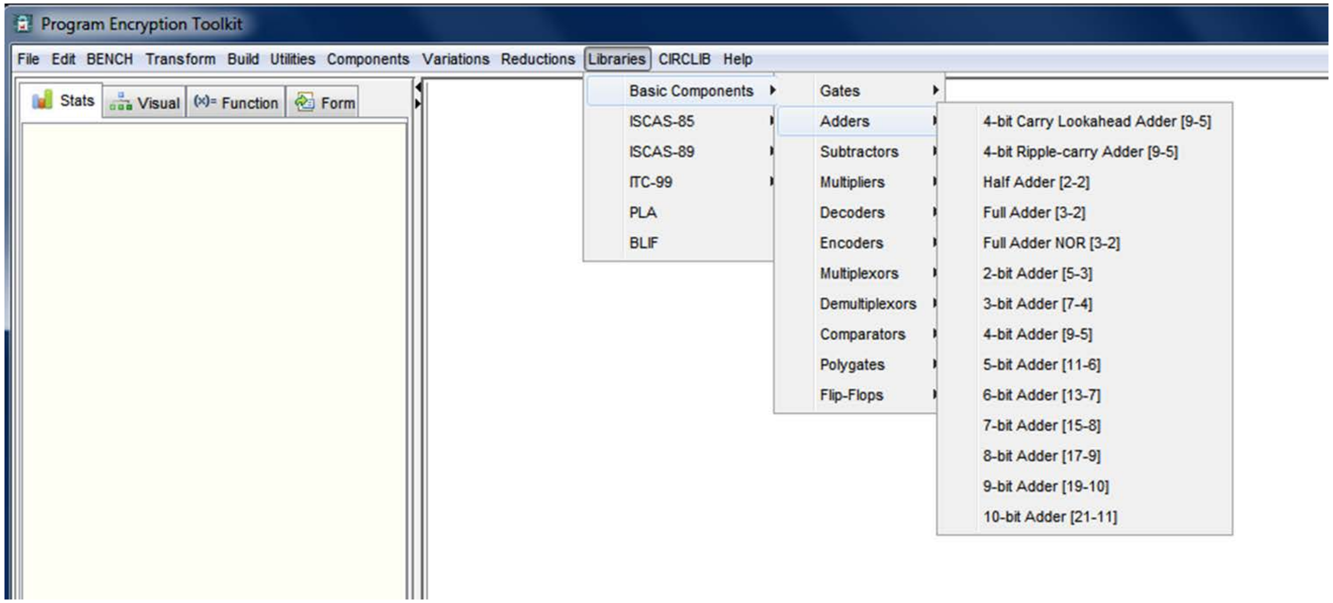
```
Number of inputs: 6
Number of outputs: 2
Number of constant1: 0
Number of constant0: 0
Intermediate gates: 7
   ANDs: 1
   ORs: 2
   XORs: 0
   NANDs: 0
   NORs: 0
   NXORs: 0
   BUFFERs: 0
   NOTs: 4
   DFF: 0
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 2
Size of Largest Level: 6
Size of Largest Intermediate Level: 5
Maximum Fan-In: 4
Maximum Fan-Out: 2
Average Fan-In: 1.7
Average Fan-Out: 1.0

  Level[0] Size =6
  Level[1] Size =5
  Level[2] Size =2
  Level[3] Size =2

  Level[0]: 0 4 3 2 1 5
  Level[1]: 6 8 10 7 9
  Level[2]: 11 12
  Level[3]: 13 14
```

```
#
# 6 inputs
# 2 outputs
# 4 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 3
# Intermediate nodes: 7
#    ANDs: 1
#    ORs: 2
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(12)
OUTPUT(11)

6=NOT(3)
7=NOT(2)
8=NOT(0)
9=NOT(1)
10=AND(4,5)
11=OR(10,1)
12=OR(8,9,7,6)
```

```
   NXORs: 0
   BUFFERs: 0
   NOTs: 4
   DFF: 0
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 2
Size of Largest Level: 6
Size of Largest Intermediate Level: 5
Maximum Fan-In: 4
Maximum Fan-Out: 2
Average Fan-In: 1.7
Average Fan-Out: 1.0
```

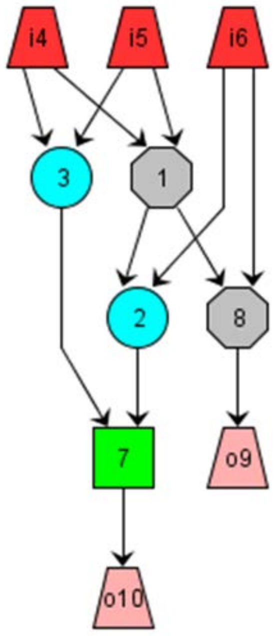
Use Case 6: Create a pre-defined BENCH component

- Libraries -> Basic Components -> …
  - Browse and choose a file to save to



```
INPUT(4)
INPUT(5)
INPUT(6)

OUTPUT(8)
OUTPUT(7)

1=XOR(4,5)
3=AND(4,5)
8=XOR(1,6)
2=AND(1,6)
7=OR(2,3)
```

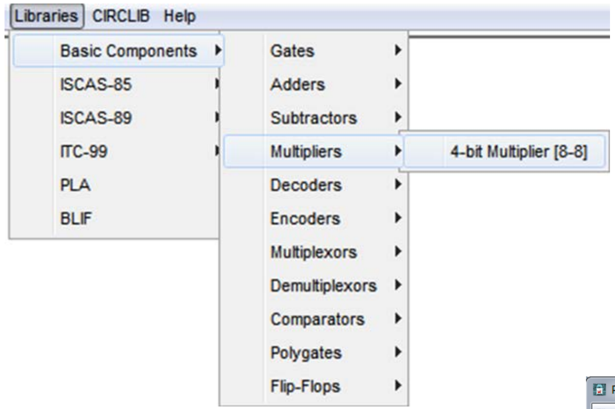
Example: Full Adder (3-2)

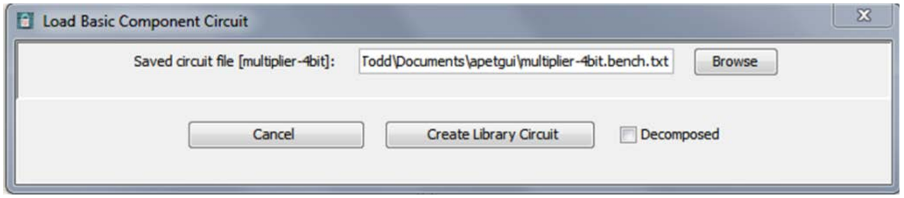


Use Case 6: Create a pre-defined BENCH component

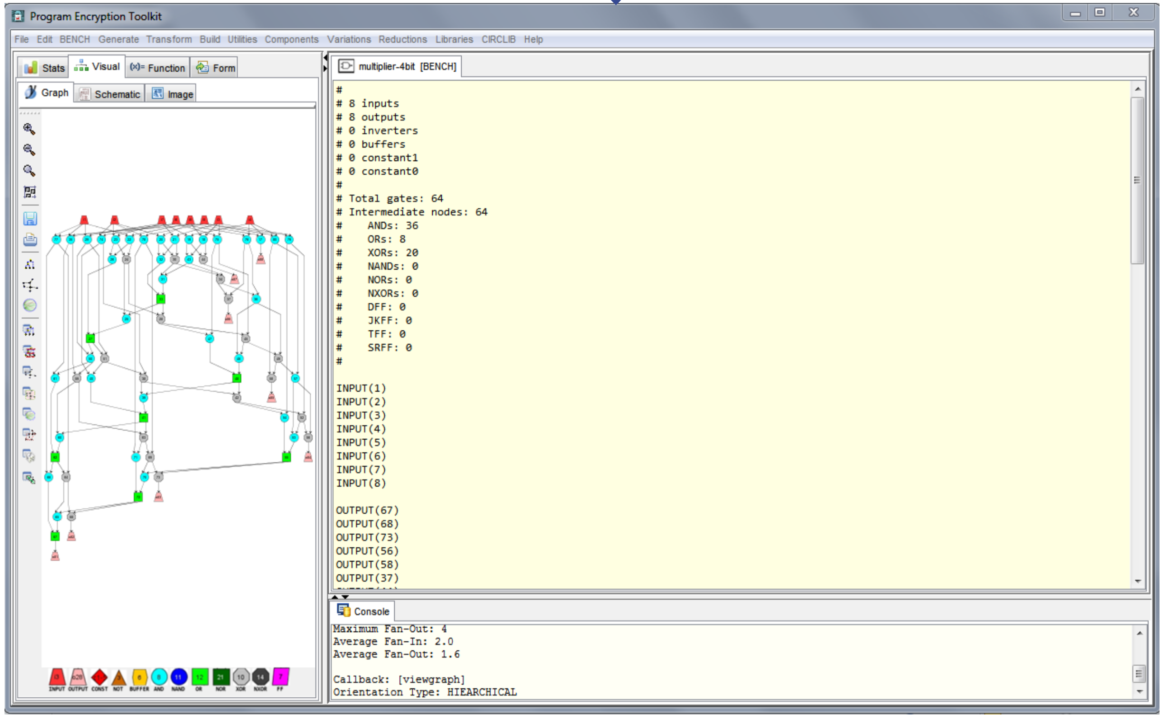
- Libraries -> Basic Components -> …

Example: 4-bit Multiplier





Load Basic Component Circuit
Saved circuit file [multiplier-4bit]:
Todd\Documents\apetgui\multiplier-4bit.bench.txt
Browse
Cancel
Create Library Circuit
Decomposed



Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats
Visual
Function
Form
Graph
Schematic
Image
multiplier-4bit [BENCH]
Console

```
#
# 8 inputs
# 8 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 64
# Intermediate nodes: 64
#    ANDs: 36
#    ORs: 8
#    XORs: 20
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)

OUTPUT(67)
OUTPUT(68)
OUTPUT(73)
OUTPUT(56)
OUTPUT(58)
OUTPUT(37)
```

```
Maximum Fan-Out: 4
Average Fan-In: 2.0
Average Fan-Out: 1.6

Callback: [viewgraph]
Orientation Type: HIEARCHICAL
```

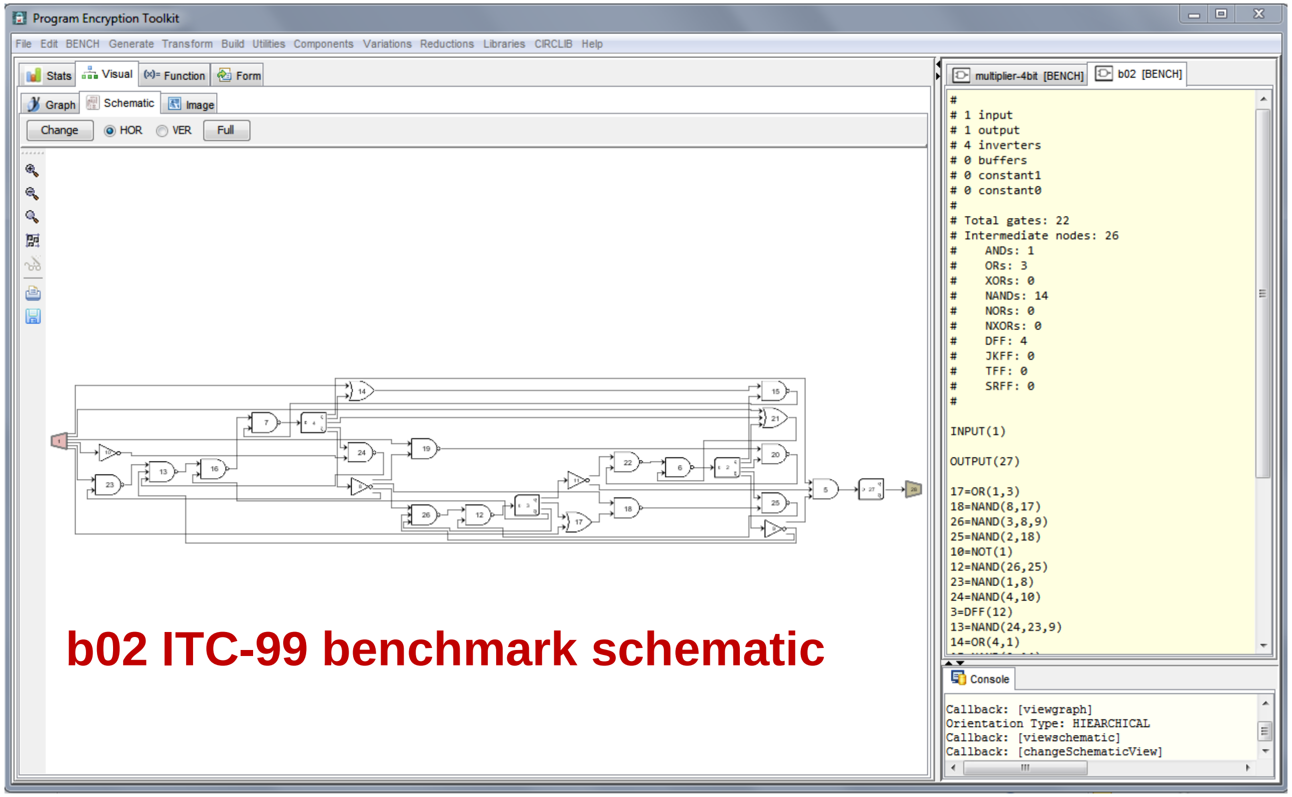
Then:
BENCH->Compile Combinational
BENCH-> …

Use Case 6: Create a pre-defined BENCH component

- **Basic Gates (2 – 4 input)**
  - AND
  - OR
  - XOR
  - NAND
  - NOR
  - NXOR
  - BUFFER
  - INVERTER
- **Adders**
- **Subtractors**
- **Multipliers**
- **Decoders**
- **Encoders**
- **Multiplexors**
- **Demultiplexors**
- **Comparators**
- **Flipflops**
- **Polygates**

Use Case 6: Create a pre-defined BENCH component

- **ISCAS-85 Benchmarks (combinational)**
- **ISCAS-89 Benchmarks (sequential)**
- **ITC-99 Benchmarks (sequential)**


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form
Graph Schematic Image
Change HOR VER Full
multiplier-4bit [BENCH] b02 [BENCH]
Console

```
#
# 1 input
# 1 output
# 4 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 22
# Intermediate nodes: 26
#    ANDs: 1
#    ORs: 3
#    XORs: 0
#    NANDs: 14
#    NORs: 0
#    NXORs: 0
#    DFF: 4
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)

OUTPUT(27)

17=OR(1,3)
18=NAND(8,17)
26=NAND(3,8,9)
25=NAND(2,18)
10=NOT(1)
12=NAND(26,25)
23=NAND(1,8)
24=NAND(4,10)
3=DFF(12)
13=NAND(24,23,9)
14=OR(4,1)
```

```
Callback: [viewgraph]
Orientation Type: HIEARCHICAL
Callback: [viewschematic]
Callback: [changeSchematicView]
```

b02 ITC-99 benchmark schematic

Use Case 7: Export a BENCH File from Text Panel in Different Formats

- File -> Export
  - BENCH ← Extended options
  - Image ← Image format
  - Truth Table ← Text format
  - Logic Friday (CSV) Truth Table
  - GraphML ← Native format for yEd
  - VHDL
  - UW ← University of Wisconsin format
  - BDD ← Image format

Major Transforms:

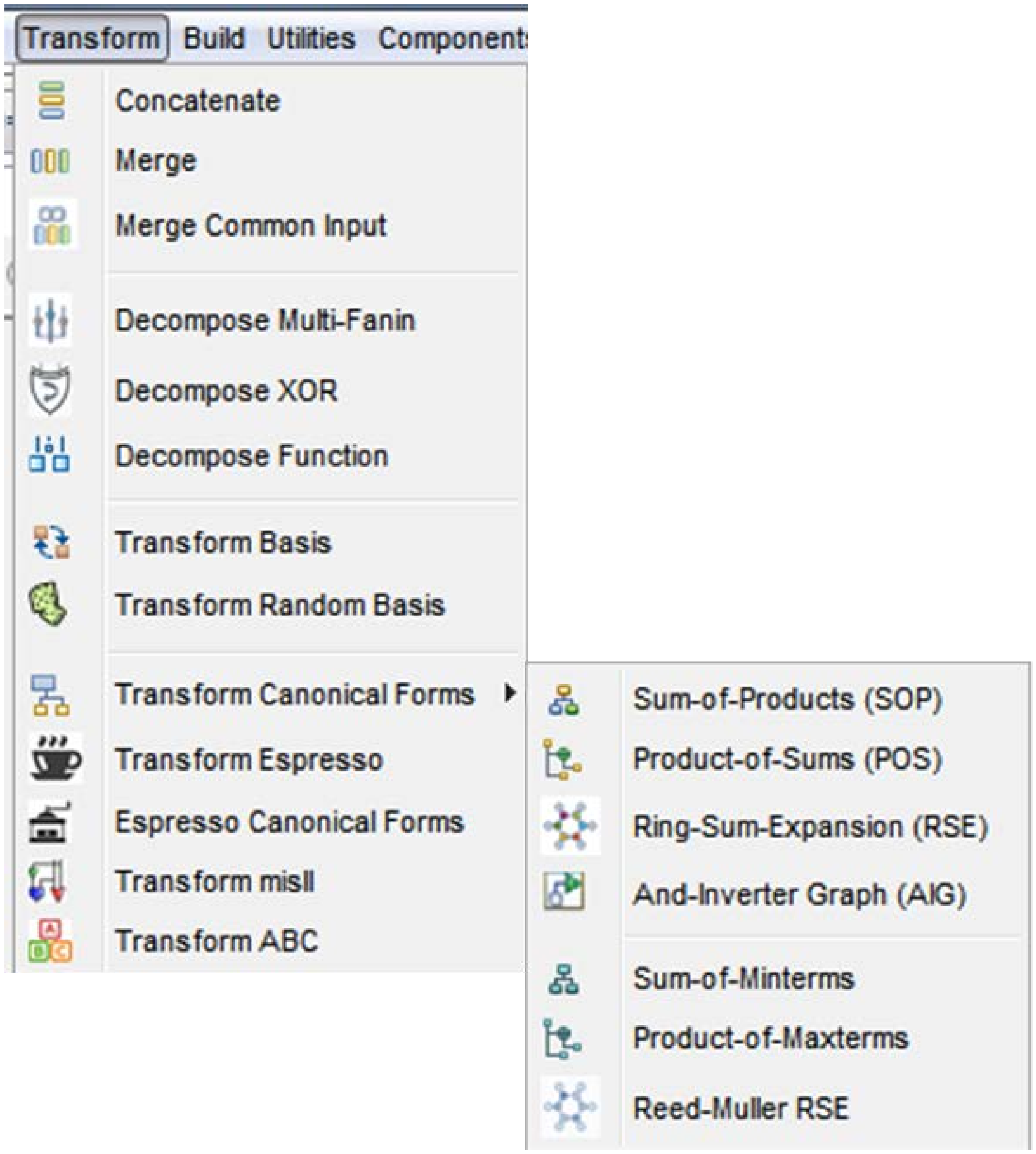
Concat
Merge
Merge Common Input

Decompose Multi-fanin
Decompose XOR
Decompose Function

Transform Basis / Random Basis

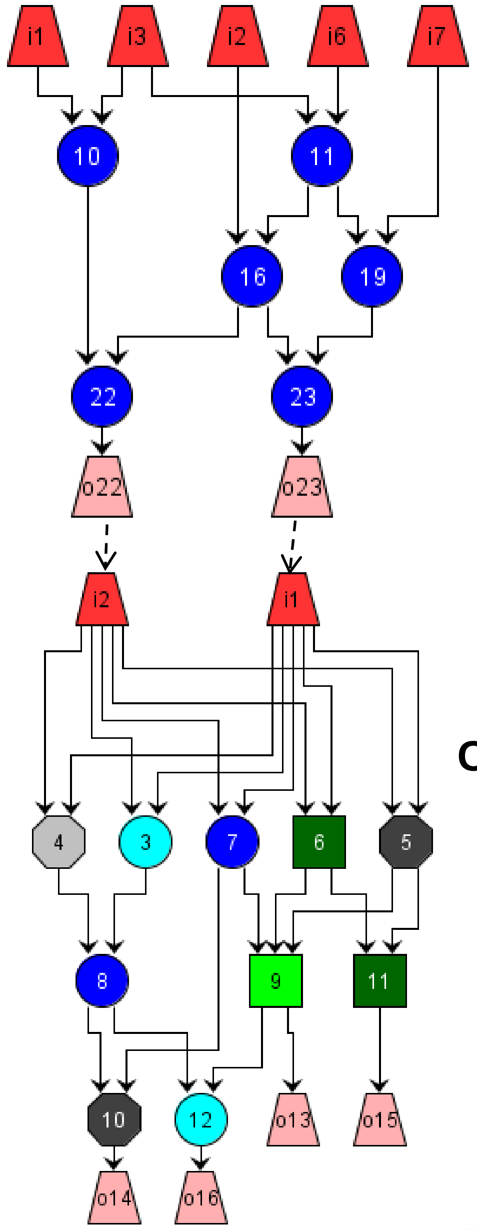
Transform SOP/POS/RSE/AIG
Transform SOM/POM/ReedMuller (reduced)
Transform Espresso / Espresso Canonical Forms
Transform misII
Transform ABC

Transforms


Transform
Build
Utilities
Component
Concatenate
Merge
Merge Common Input
Decompose Multi-Fanin
Decompose XOR
Decompose Function
Transform Basis
Transform Random Basis
Transform Canonical Forms
Transform Espresso
Espresso Canonical Forms
Transform misII
Transform ABC
Sum-of-Products (SOP)
Product-of-Sums (POS)
Ring-Sum-Expansion (RSE)
And-Inverter Graph (AIG)
Sum-of-Minterms
Product-of-Maxterms
Reed-Muller RSE

Transforms: Concatenate

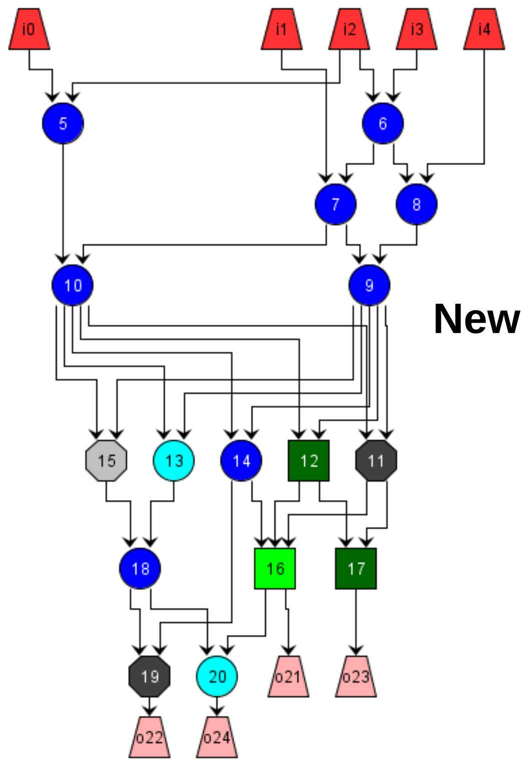
Original (A)

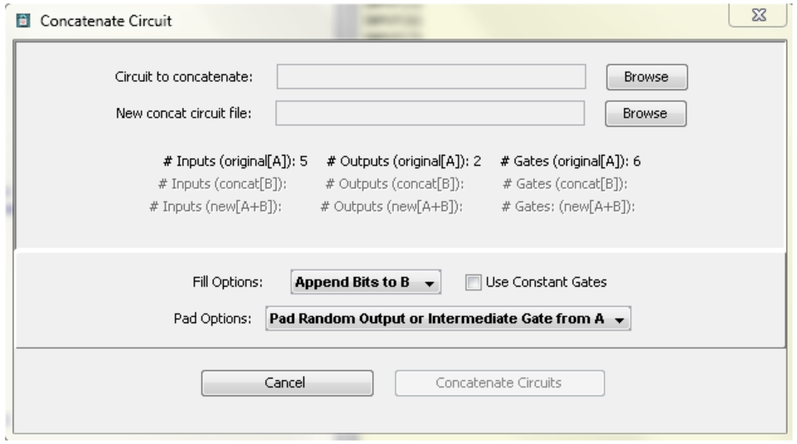


Example:
Ideal concatenate where
# of outputs = # of inputs

Circuit to Concatenate (B)

New Concat Circuit

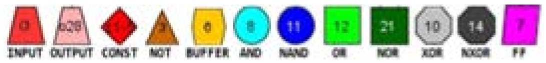



Concatenate Circuit

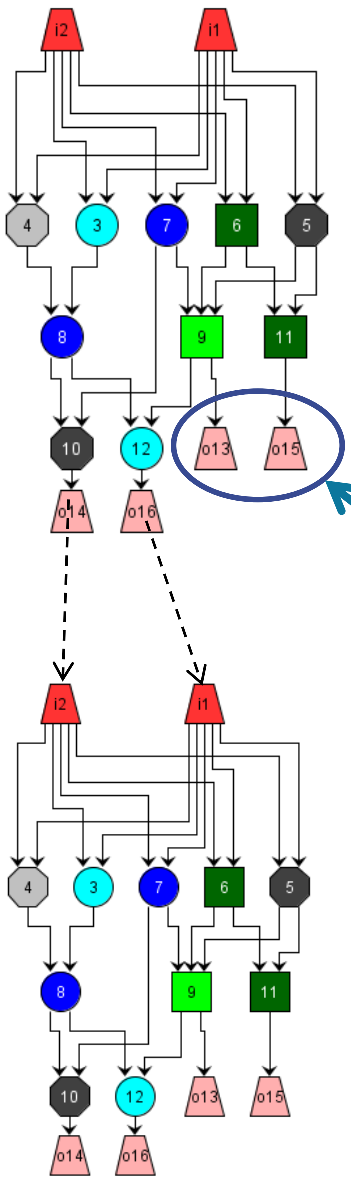
Circuit to concatenate:
New concat circuit file:

Inputs (original[A]): 5 # Outputs (original[A]): 2 # Gates (original[A]): 6
Inputs (concat[B]): # Outputs (concat[B]): # Gates (concat[B]):
Inputs (new[A+B]): # Outputs (new[A+B]): # Gates: (new[A+B]):

Fill Options: Append Bits to B
Use Constant Gates
Pad Options: Pad Random Output or Intermediate Gate from A

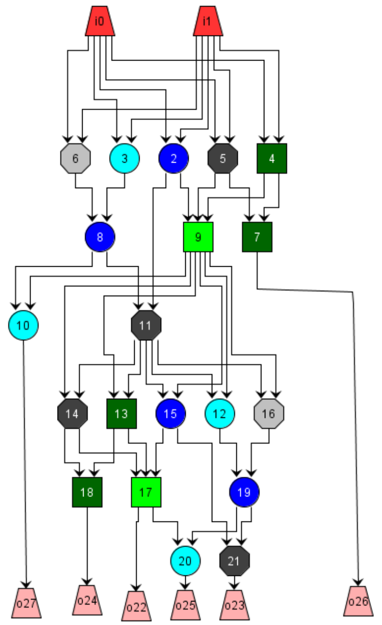


Transforms: Concatenate

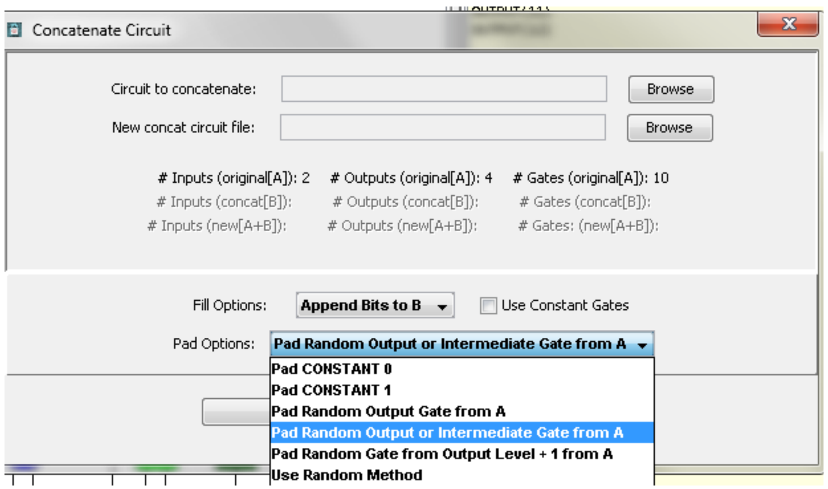


Original (A)

Example:
Created with Pad Option = Use Random Method



What to do with these 2 outputs?
Specify Pad Options



Concatenate Circuit

Circuit to concatenate: Browse

New concat circuit file: Browse

Inputs (original[A]): 2 # Outputs (original[A]): 4 # Gates (original[A]): 10
Inputs (concat[B]): # Outputs (concat[B]): # Gates (concat[B]):
Inputs (new[A+B]): # Outputs (new[A+B]): # Gates: (new[A+B]):

Fill Options: Append Bits to B ☐ Use Constant Gates

Pad Options: Pad Random Output or Intermediate Gate from A

- Pad CONSTANT 0
- Pad CONSTANT 1
- Pad Random Output Gate from A
- Pad Random Output or Intermediate Gate from A
- Pad Random Gate from Output Level + 1 from A
- Use Random Method

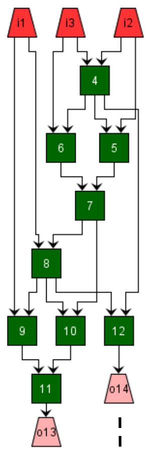
New Concat Circuit

Circuit to Concatenate (B)

INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

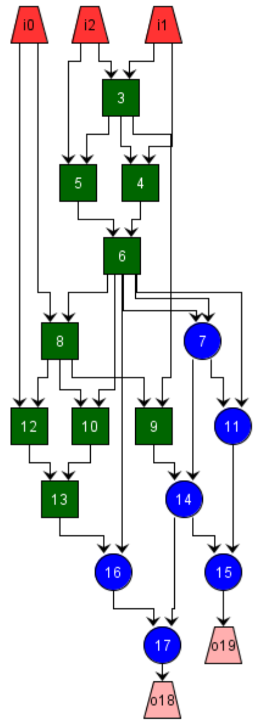
Transforms: Concatenate

Original (A)



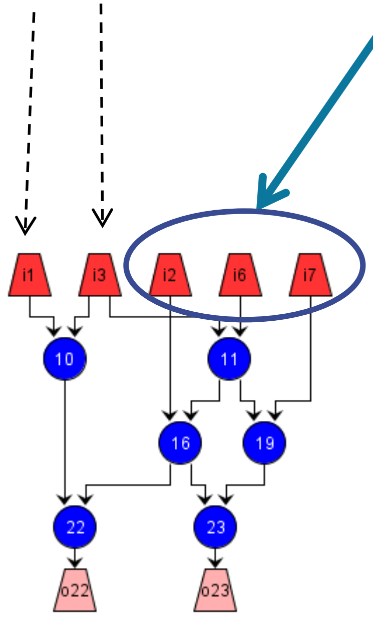
Example:
Created with Pad Option =
Pad Random Gate from Output Level + 1 from A

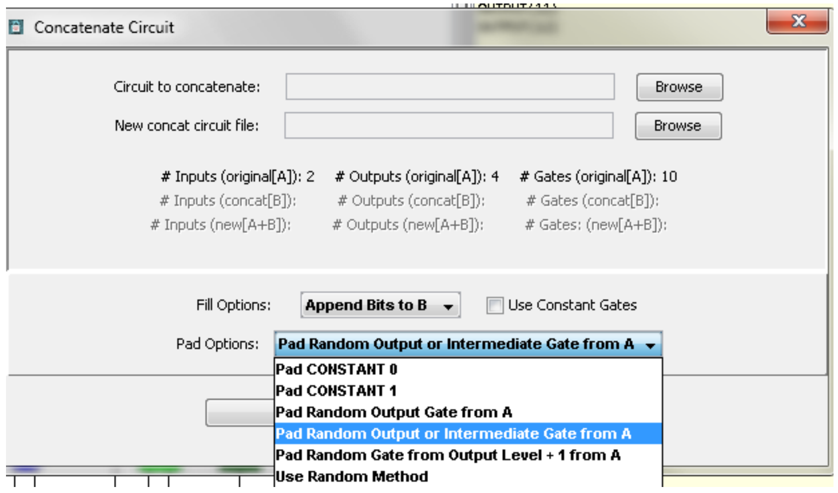
Fill Options: Append Bits to B



New Concat Circuit

What to do with these 3 inputs?
Specify Fill and Pad Options





Concatenate Circuit

Circuit to concatenate: Browse
New concat circuit file: Browse

# Inputs (original[A]): 2 # Outputs (original[A]): 4 # Gates (original[A]): 10
# Inputs (concat[B]): # Outputs (concat[B]): # Gates (concat[B]):
# Inputs (new[A+B]): # Outputs (new[A+B]): # Gates: (new[A+B]):

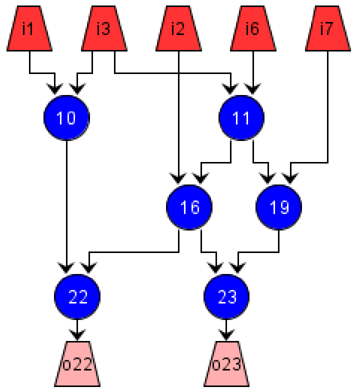
Fill Options: Append Bits to B ☐ Use Constant Gates
Pad Options: Pad Random Output or Intermediate Gate from A

Pad CONSTANT 0
Pad CONSTANT 1
Pad Random Output Gate from A
Pad Random Output or Intermediate Gate from A
Pad Random Gate from Output Level + 1 from A
Use Random Method

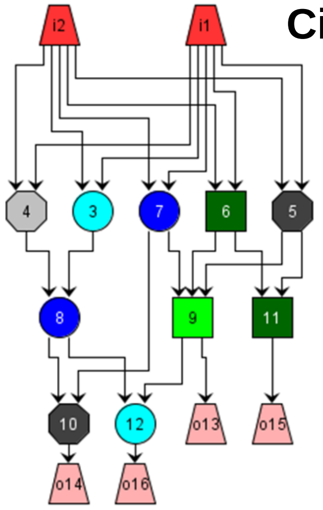
Circuit to Concatenate (B)

Transforms: Merge

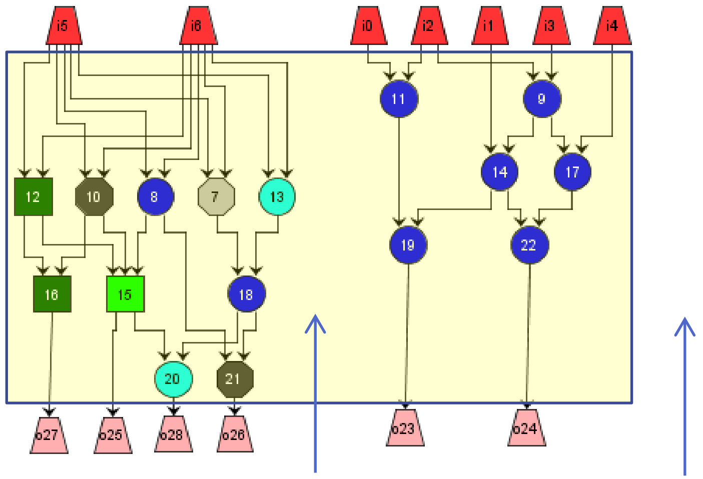
Original (A)



Circuit to Merge (B)

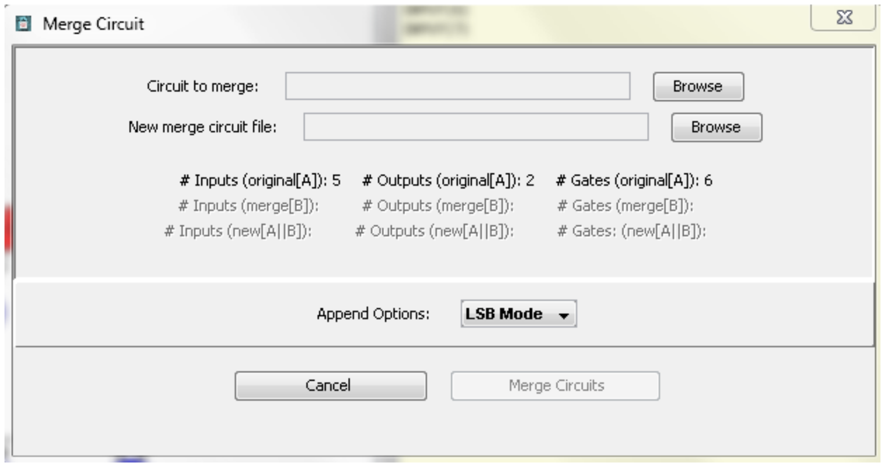


New Merge Circuit



MSB Append

LSB Append



Merge Circuit

Circuit to merge: Browse

New merge circuit file: Browse

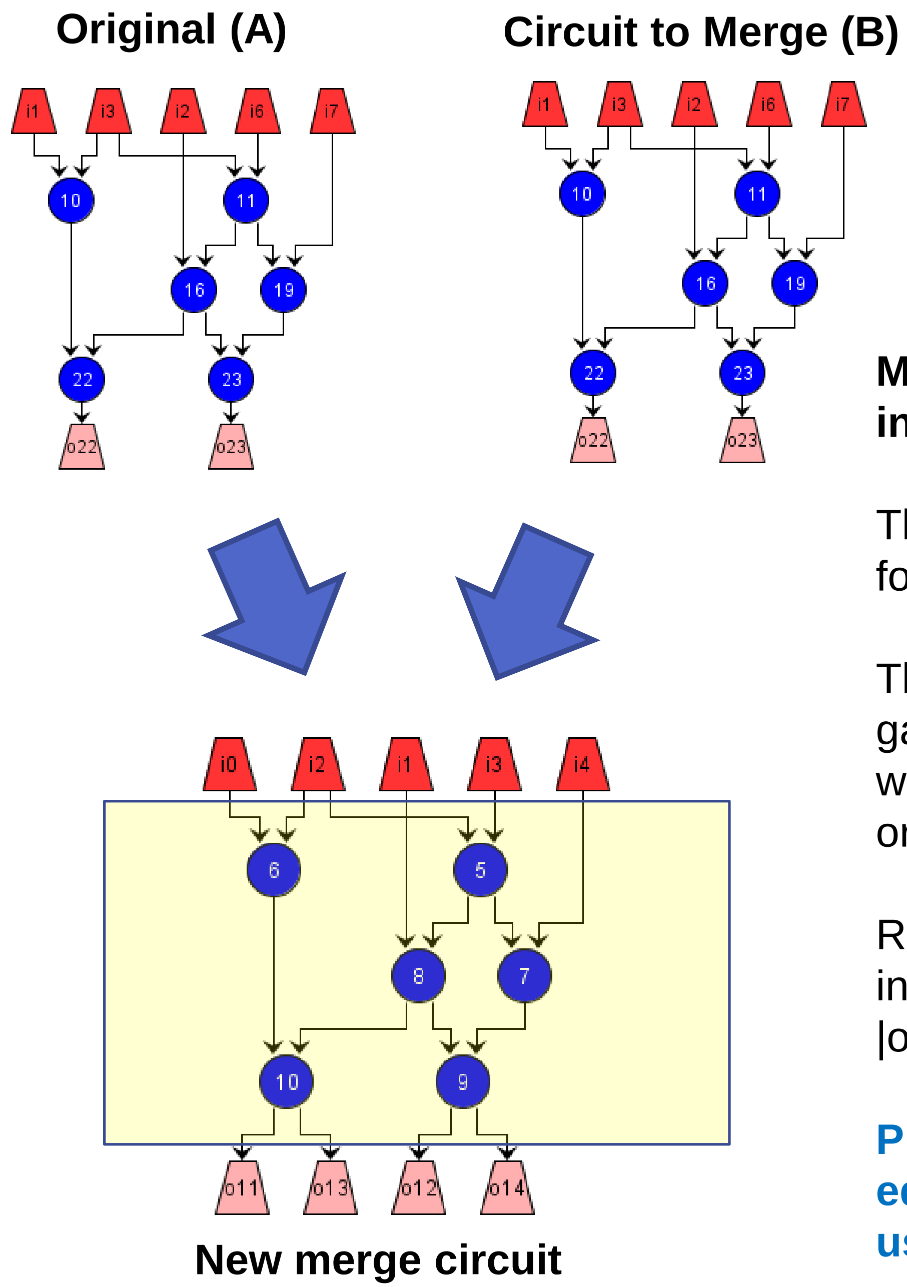
Inputs (original[A]): 5 # Outputs (original[A]): 2 # Gates (original[A]): 6
Inputs (merge[B]): # Outputs (merge[B]): # Gates (merge[B]):
Inputs (new[A||B]): # Outputs (new[A||B]): # Gates: (new[A||B]):

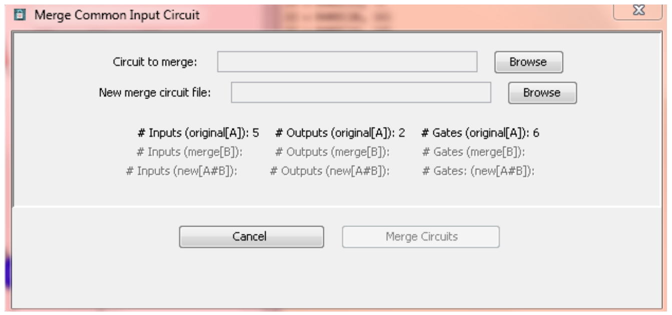
Append Options: LSB Mode

Cancel Merge Circuits

INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Transforms: Merge Common Input





Circuit to merge:
New merge circuit file:

Inputs (original[A]): 5 # Outputs (original[A]): 2 # Gates (original[A]): 6
Inputs (merge[B]): # Outputs (merge[B]): # Gates (merge[B]):
Inputs (new[A#B]): # Outputs (new[A#B]): # Gates: (new[A#B]):

Merges two circuits with the same # of inputs:

The inputs are assumed to be symmetrical for both circuits (A and B)

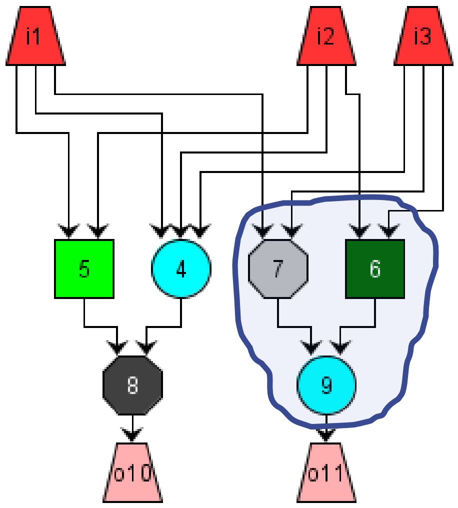
The merge attempts to match the fan-in of gates and gate types of the circuit to merge with the fan-in gates and gate types of the original

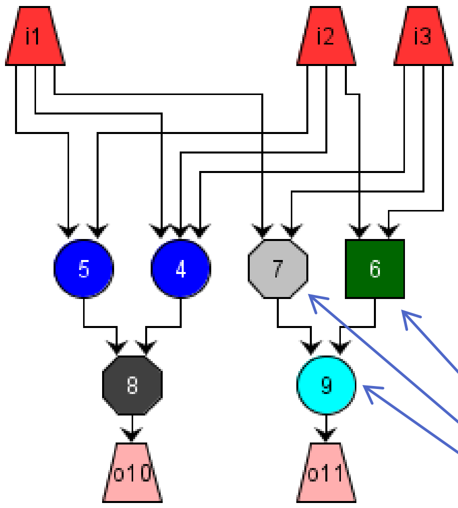
Resulting circuit will have the same # of inputs and output size equal to |outputs A| + |outputs B|

Produces a circuit with functionally equivalent outputs as that of A and B, using the same inputs space as A and B

Transforms: Merge Common Input

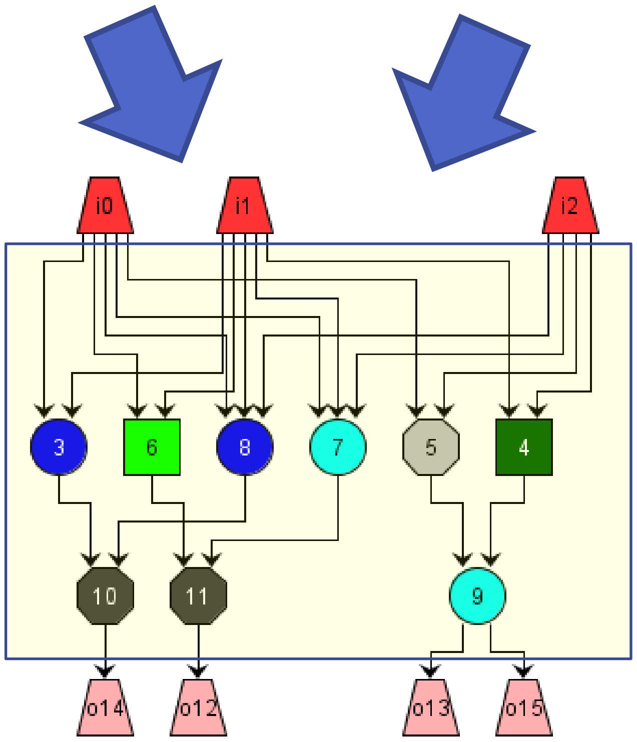
Original (A)





Circuit to Merge (B)

These gates have identical type and input fan-in as those of the original (A)

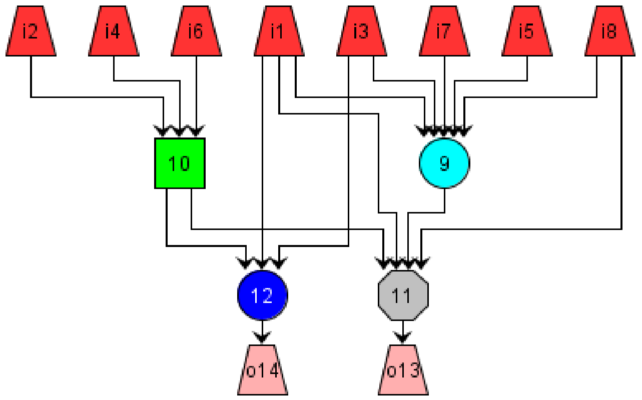


New merge circuit

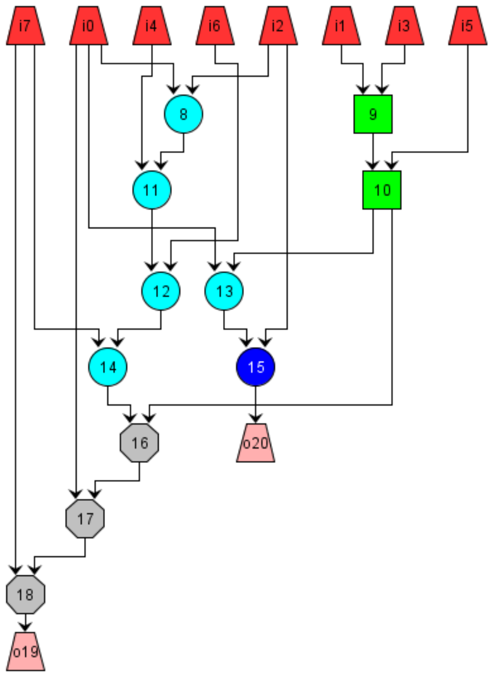
New circuit is renumbered…

INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Transforms: Decompose Multi-Fanin



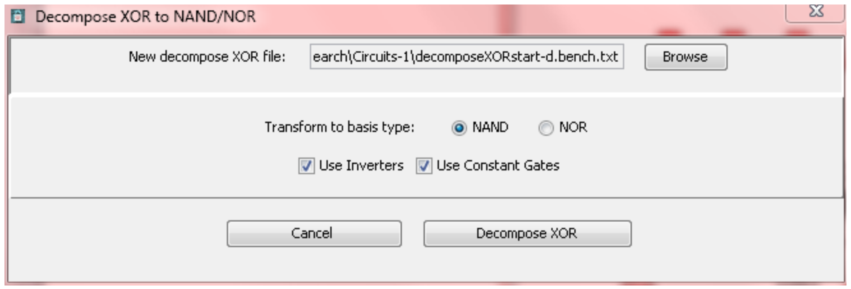
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)

OUTPUT(11)
OUTPUT(12)

9 = AND(1,3,5,7,8)
10 = OR(2,4,6)
11 = XOR(9,10,1,8)
12 = NAND(10,1,3)
```



```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)

OUTPUT(18)
OUTPUT(15)

8=AND(0,2)
9=OR(1,3)
10=OR(5,9)
11=AND(4,8)
12=AND(6,11)
13=AND(10,0)
14=AND(7,12)
15=NAND(2,13)
16=XOR(14,10)
17=XOR(0,16)
18=XOR(7,17)
```

INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

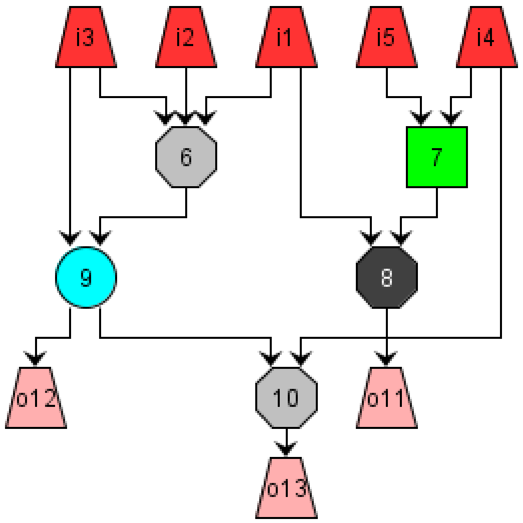
Transforms: Decompose XOR

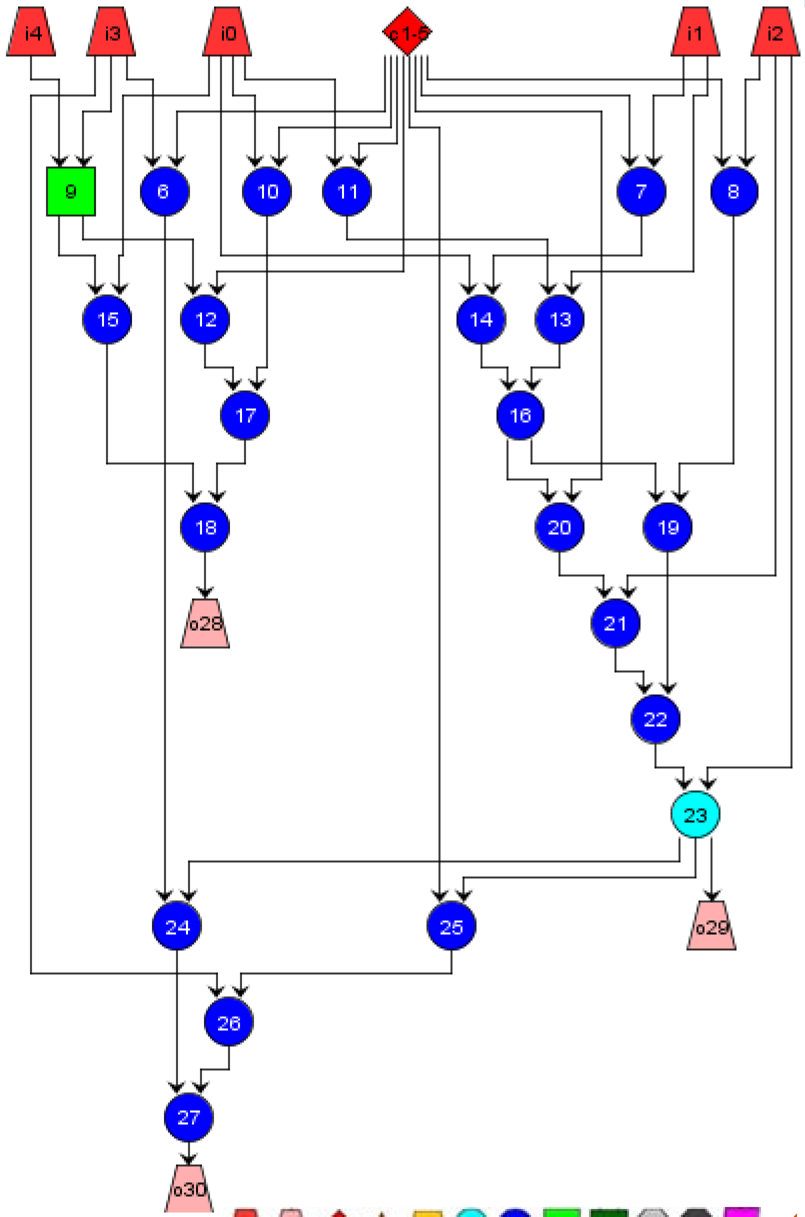

Decompose XOR to NAND/NOR

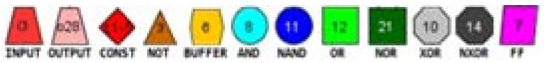
New decompose XOR file: earch\Circuits-1\decomposeXORstart-d.bench.txt

Transform to basis type: [✓ NAND] NOR

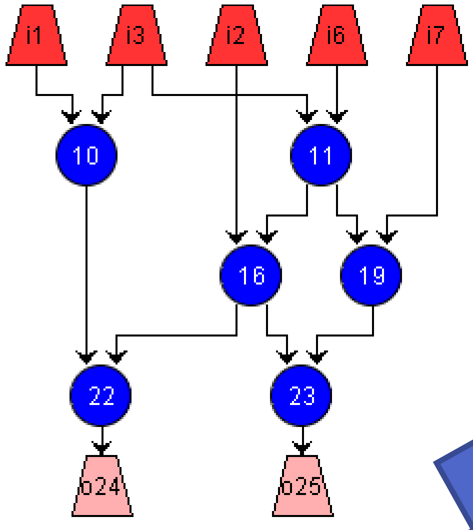
[✓ Use Inverters] [✓ Use Constant Gates]

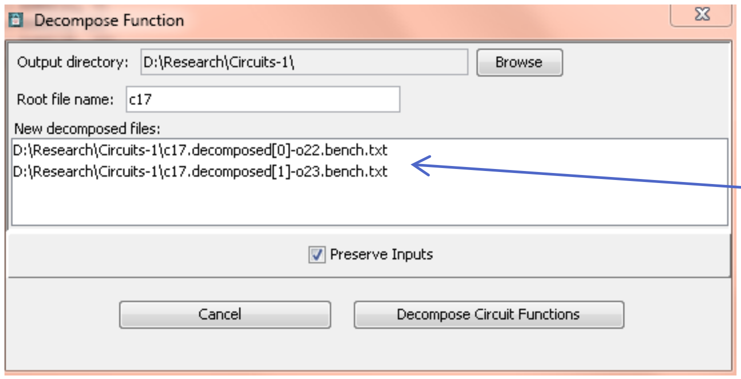






Transforms: Decompose Function




Decompose Function

Output directory: D:\Research\Circuits-1\

Root file name: c17

New decomposed files:

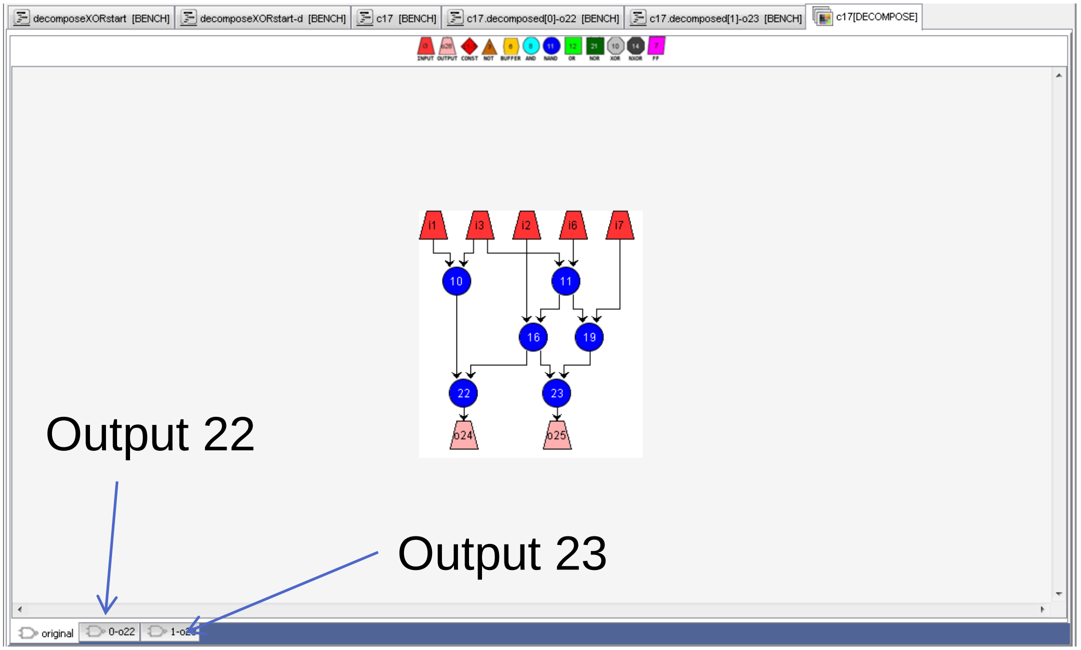
D:\Research\Circuits-1\c17.decomposed[0]-o22.bench.txt
D:\Research\Circuits-1\c17.decomposed[1]-o23.bench.txt

[✓] Preserve Inputs

Decomposed file names are automatically assigned based on circuit name

Decomposes a circuit by functional output: one circuit is produced for each output, keeping appropriate gates from original circuit

To keep the original number of inputs, check "Preserve Inputs"

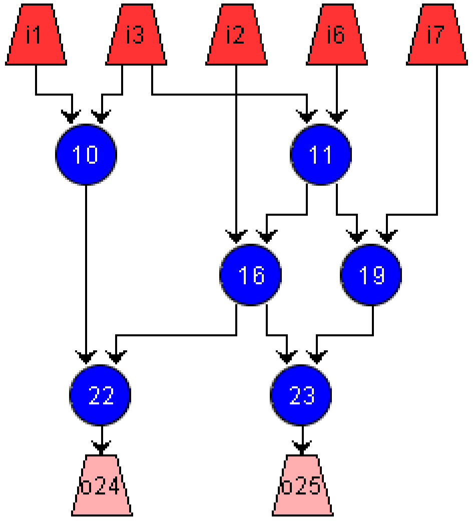


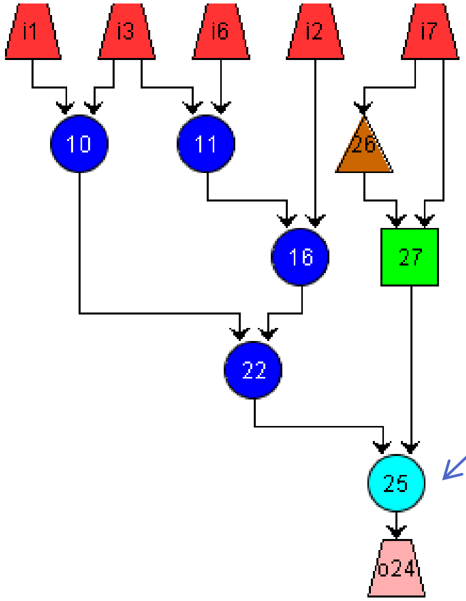
Output 22

Output 23

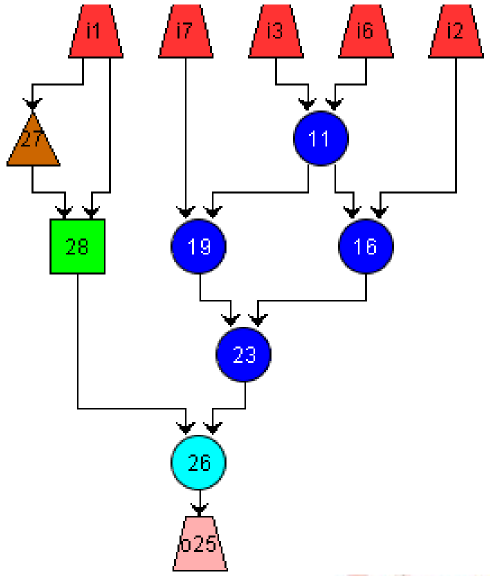
Transforms: Decompose Function

With “Preserve Inputs”





Generates logically redundant gates…



Generates logically redundant gates…

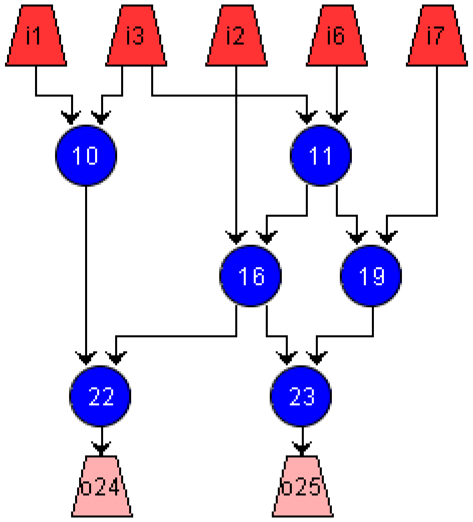
Decomposes a circuit by functional output: one circuit is produced for each output, keeping appropriate gates from original circuit

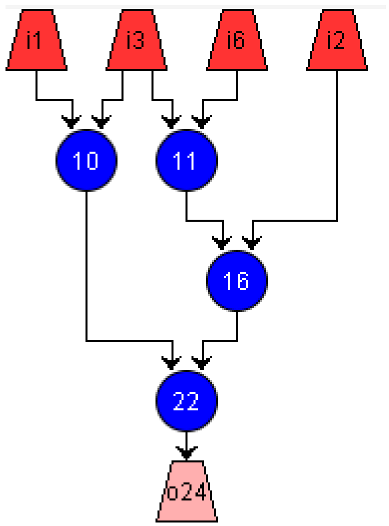
To keep the original number of inputs, check “Preserve Inputs”

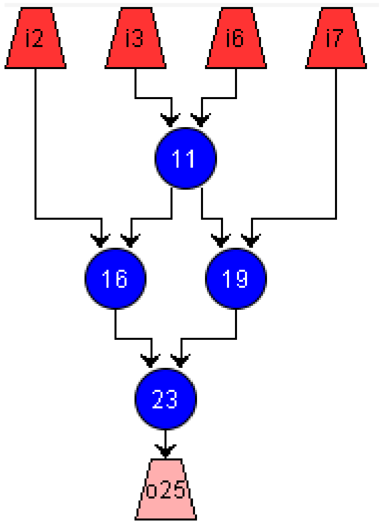
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Transforms: Decompose Function

Without “Preserve Inputs”





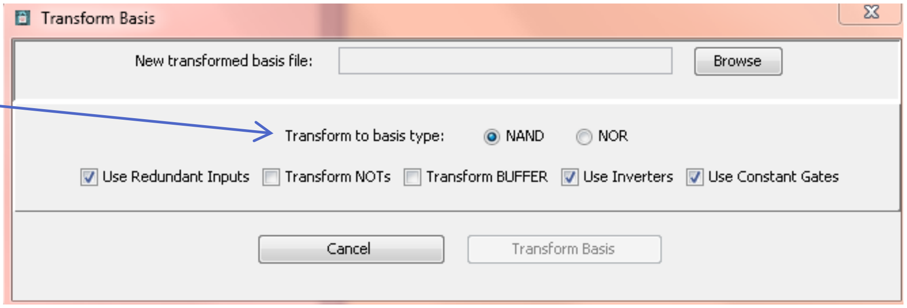


Decomposes a circuit by functional output: one circuit is produced for each output, keeping appropriate gates from original circuit

To keep the original number of inputs, check “Preserve Inputs”

Transforms: Transform Basis

Transforms each gate into a NAND-only or NOR-only representation:


Transform Basis

New transformed basis file: Browse

Transform to basis type: [✓ NAND] NOR

[✓ Use Redundant Inputs] Transform NOTs Transform BUFFER [✓ Use Inverters] [✓ Use Constant Gates]

Cancel Transform Basis

For NAND-only, NAND gates are obviously left unchanged and the same applies for NOR-only and NOR gates

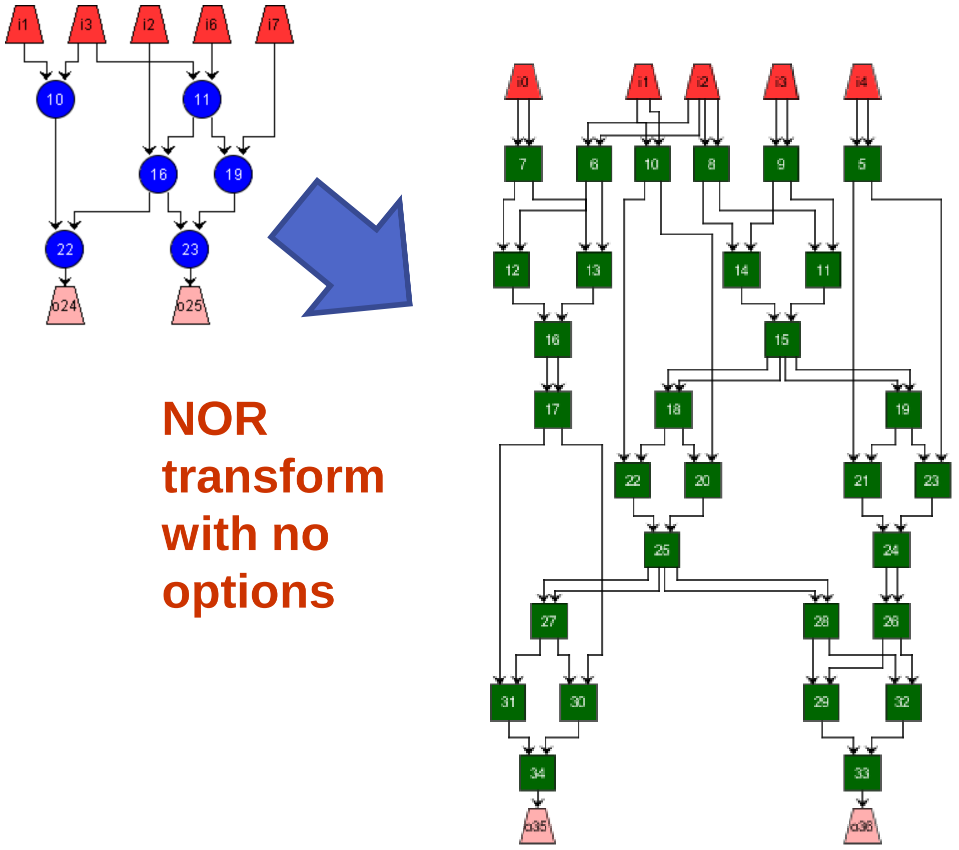
Options:

Use Redundant Input
Use Inverters
Use Constant Gates

These options allow for inverters or constant gates to be generated in the transform. Redundant inputs means that a gate can have more than one fan-in from a predecessor gate.

Transform NOTs: means that NOT gates will be transformed into equivalent NAND- or NOR-only forms

Transform BUFFERs: means that BUFFERS will be transformed into an equivalent NAND- or NOR-only form

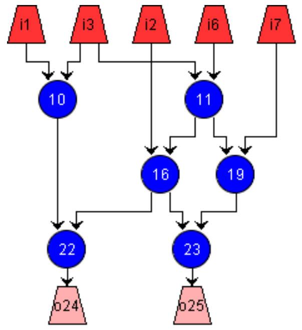


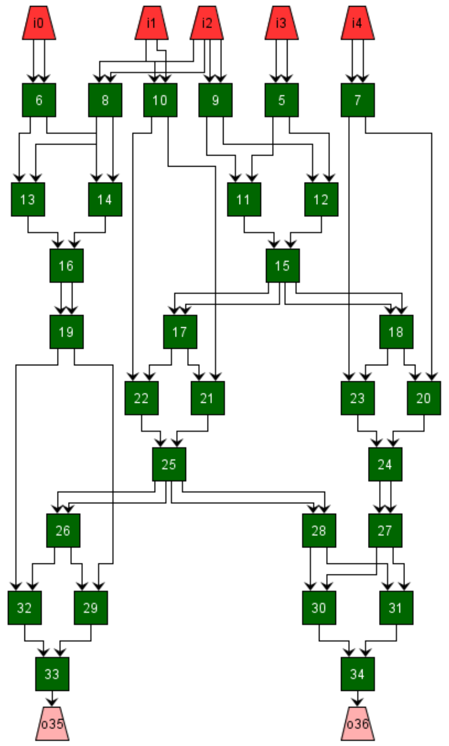
NOR transform with no options

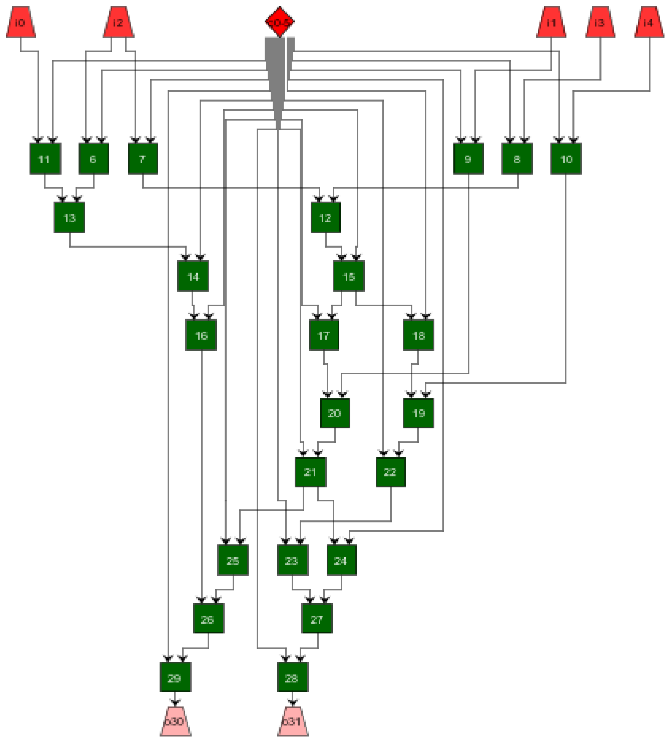
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

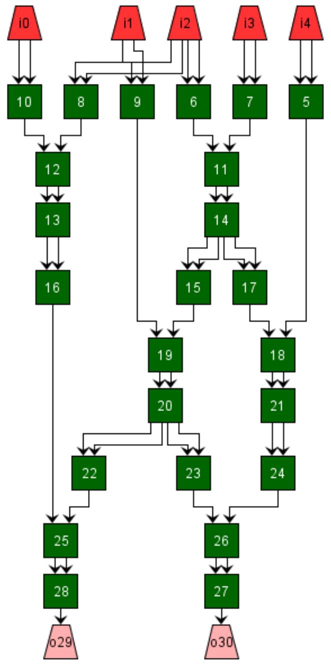
Transforms: Transform Basis

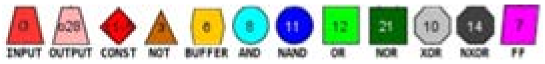
NOR transform with various options





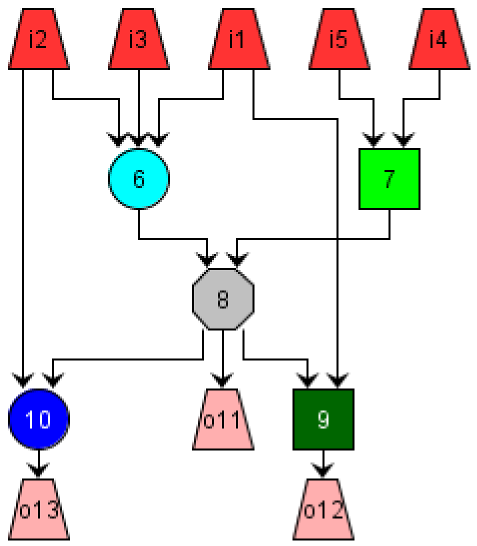




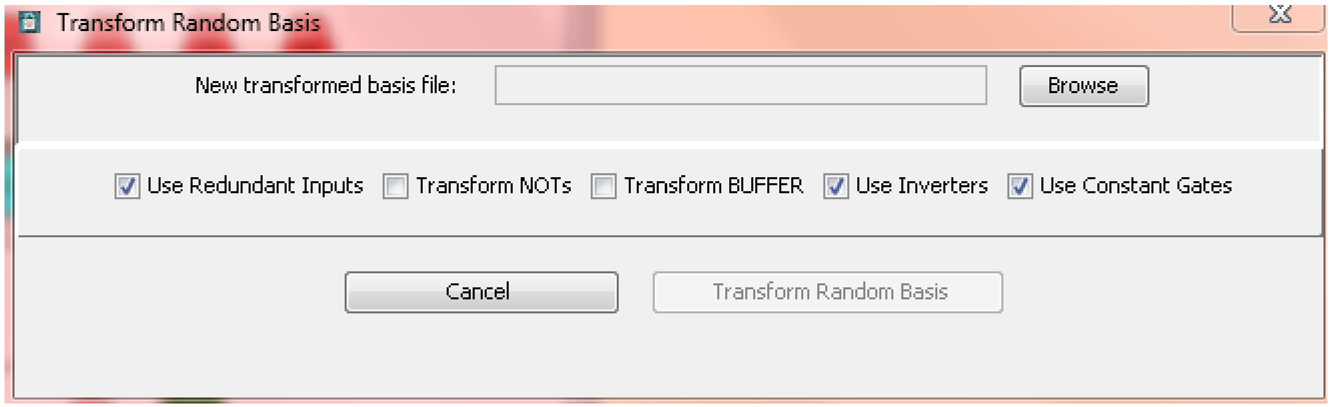


Transforms: Transform Random Basis

Transforms each gate into a NAND-only or NOR-only representation, but choose randomly which transform to use…



```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(8)
OUTPUT(9)
OUTPUT(10)

6 = AND(1,2,3)
7 = OR(4,5)
8 = XOR(6,7)
9 = NOR(1,8)
10 = NAND(2,8)
```


Transform Random Basis

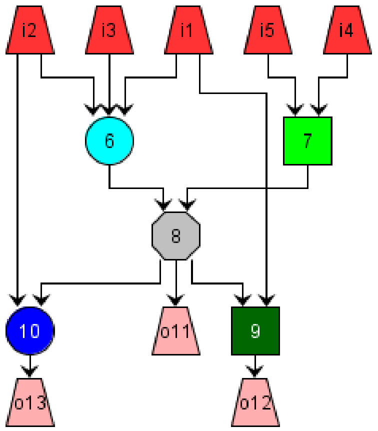
New transformed basis file: Browse

[✓] Use Redundant Inputs [ ] Transform NOTs [ ] Transform BUFFER [✓] Use Inverters [✓] Use Constant Gates

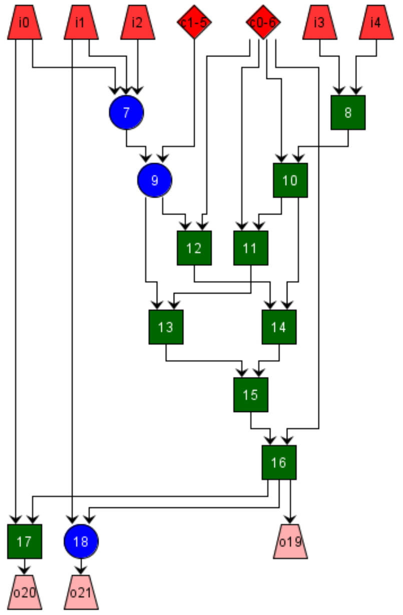
Cancel Transform Random Basis

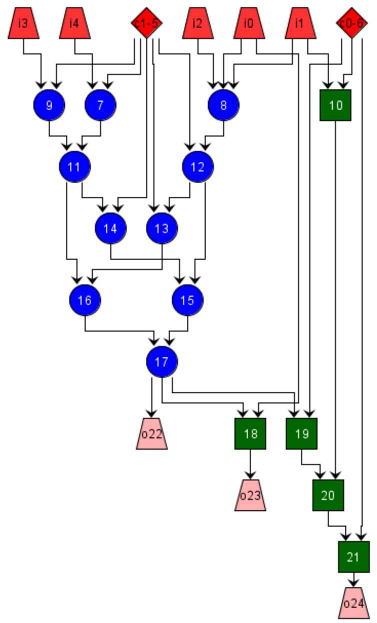
Transforms: Transform Random Basis

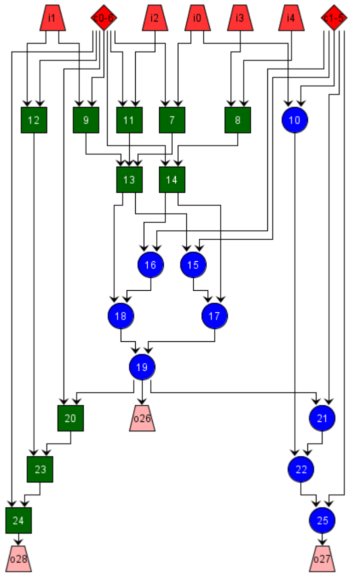
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(8)
OUTPUT(9)
OUTPUT(10)

6 = AND(1,2,3)
7 = OR(4,5)
8 = XOR(6,7)
9 = NOR(1,8)
10 = NAND(2,8)
```



Random basis transforms



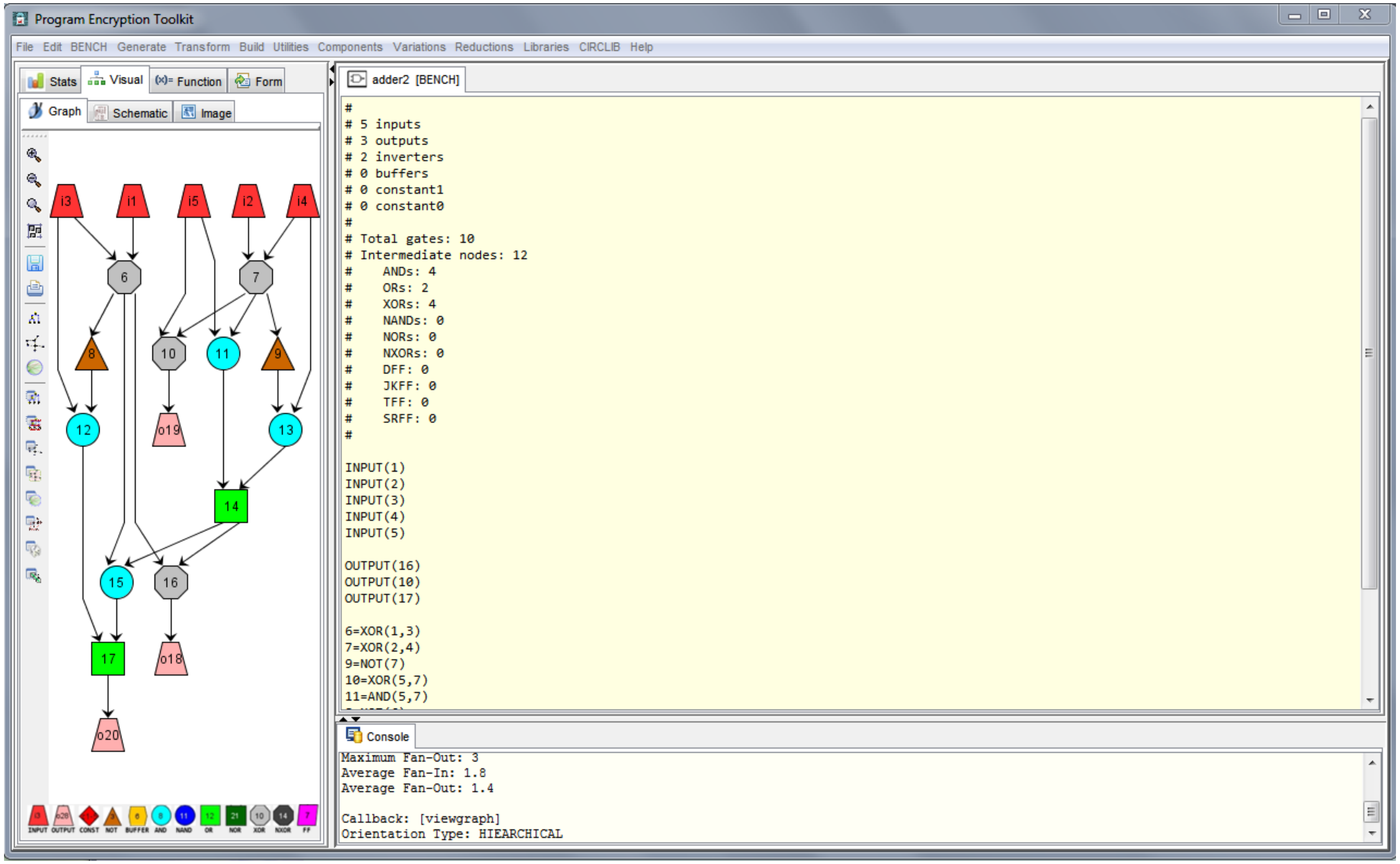




INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Example: 2-bit adder

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form
Graph Schematic Image
adder2 [BENCH]
Console
i3 i1 i5 i2 i4
6 7 8 10 11 9 12 o19 13 14 15 16 17 o18 o20
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF

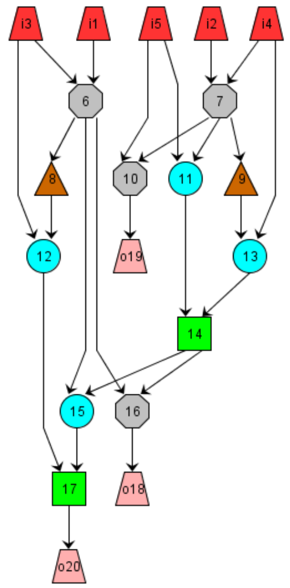
```
#
# 5 inputs
# 3 outputs
# 2 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 10
# Intermediate nodes: 12
#    ANDs: 4
#    ORs: 2
#    XORs: 4
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
```

```
Maximum Fan-Out: 3
Average Fan-In: 1.8
Average Fan-Out: 1.4

Callback: [viewgraph]
Orientation Type: HIEARCHICAL
```

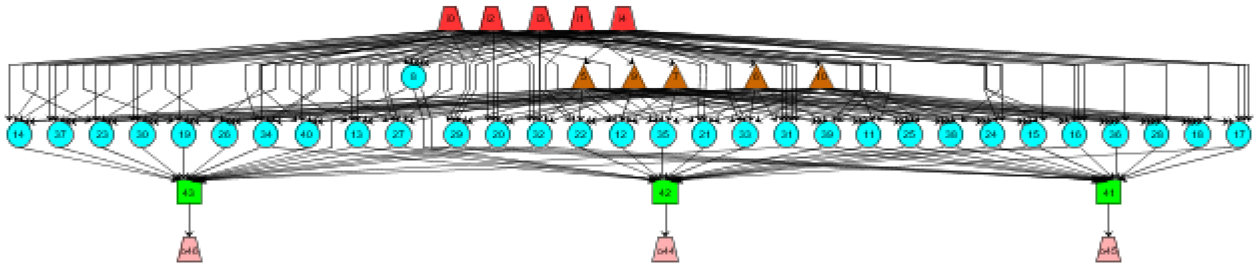
Transforms: Sum of Products

Transforms a circuit into its Sum-of-Products (unreduced) equivalent 2-level circuit representation

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)

OUTPUT(42)
OUTPUT(41)
OUTPUT(43)

5=NOT(4)
6=NOT(0)
7=NOT(3)
8=AND(0,1,2,3,4)
9=NOT(1)
10=NOT(2)
11=AND(6,1,2,7,5)
12=AND(6,9,2,7,5)
. . .
. . .
. . .
39=AND(6,9,2,3,5)
40=AND(6,1,2,7,4)
41=OR(18,36,38,39,28,17,11,31,24,25,20,32,15,16,29,8)
42=OR(21,12,38,39,33,22,17,11,35,24,25
```



```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```

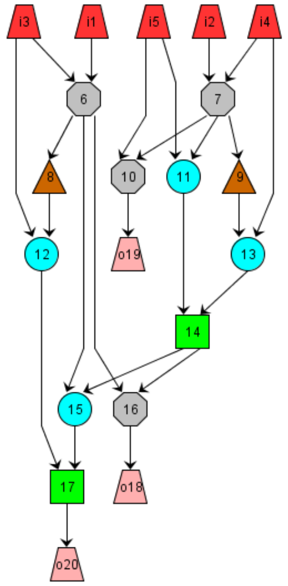


INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

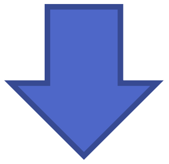
Transforms: Product of Sums

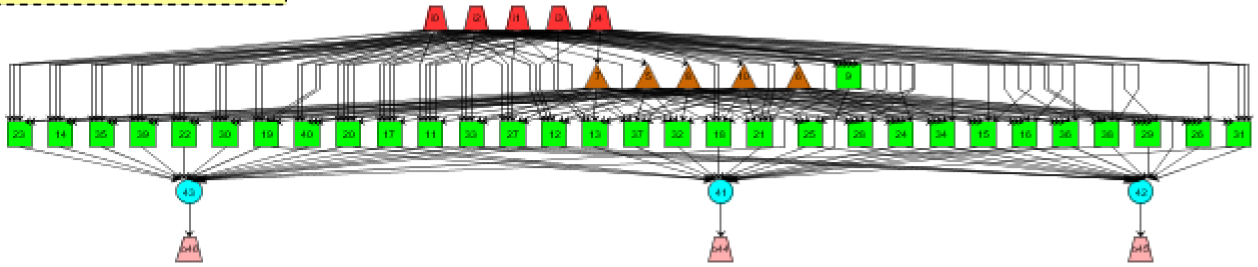
Transforms a circuit into its Product-of-Sums (unreduced) equivalent 2-level circuit representation

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```



```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)

OUTPUT(41)
OUTPUT(42)
OUTPUT(43)

5=NOT(1)
6=NOT(0)
7=NOT(4)
8=NOT(3)
9=OR(0,1,2,3,4)
10=NOT(2)
11=OR(0,1,10,3,4)
12=OR(0,5,2,8,4)
13=OR(6,1,10,3,7)
. . .
. . .
. . .
39=OR(6,5,2,3,4)
40=OR(0,5,2,8,7)
41=AND(9,19,17,15,20,25,36,37,34,31,13,21,16,24,32,18)
42=AND(9,27,11,15,33,12,25,36,38,34,31,29,16,24,28,26)
43=AND(9,19,17,27,11,14,23,20,33,12,40,30,38,22,35,39)
```

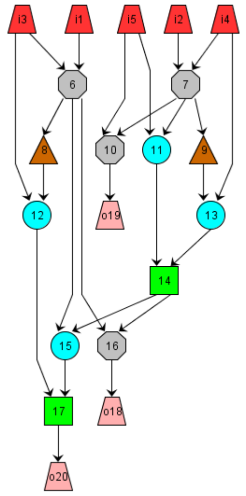




INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

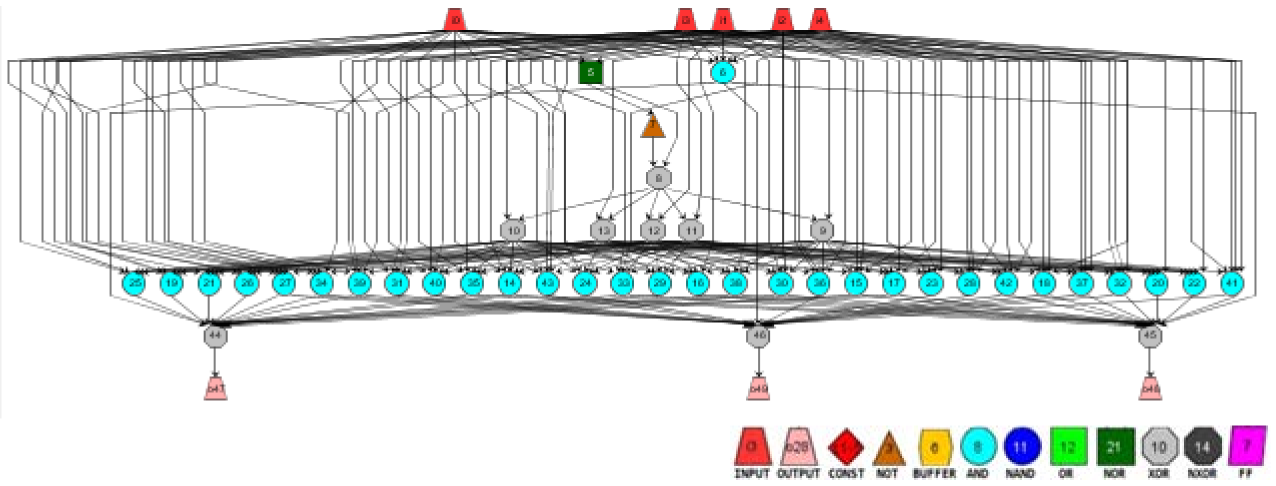
Transforms: Ring Sum Expansion

Transforms a circuit into its Ring Sum Expansion [RSE] (unreduced) equivalent 2-level circuit representation



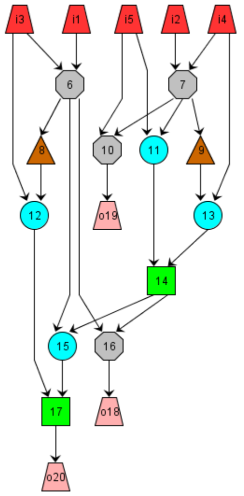
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)

OUTPUT(44)
OUTPUT(45)
OUTPUT(46)

5=NOR(0,2)
6=AND(0,1,2,3,4)
7=NOT(5)
8=XOR(5,7)
9=XOR(0,8)
10=XOR(2,8)
11=XOR(3,8)
12=XOR(1,8)
13=XOR(4,8)
14=AND(0,1,10,11,13)
15=AND(9,1,10,3,4)
. . .
. . .
. . .
42=AND(0,12,2,3,13)
43=AND(0,1,10,11,4)
44=XOR(19,29,17,38,26,25,15,30,21,33,35,34,14,39,27,6)
45=XOR(22,16,17,38,20,15,30,41,33,35,32,42,14,18,37,6)
46=XOR(28,23,36,41,40,24,32,42,34,43,31,18,37,39,27,6)
```

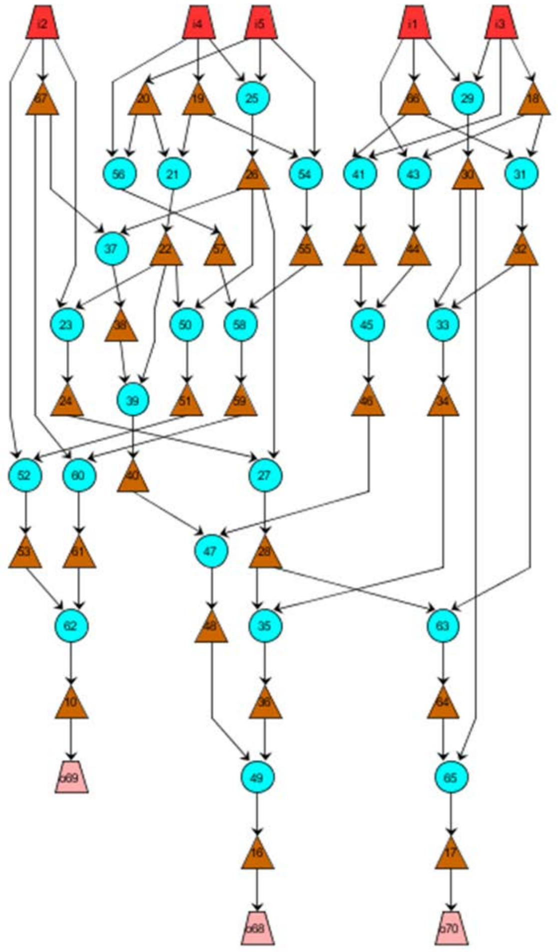


Transforms: And Inverter Graph

Transforms a circuit into one possible And-Inverter Graph [AIG] (unreduced) equivalent representation



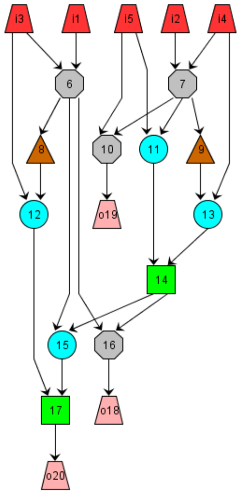
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```

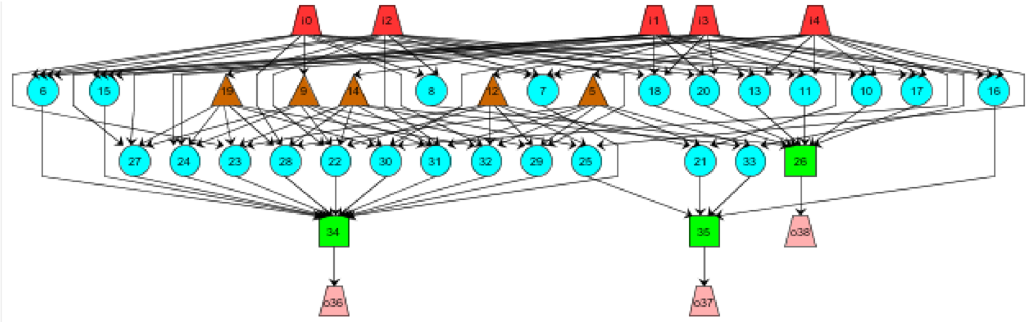


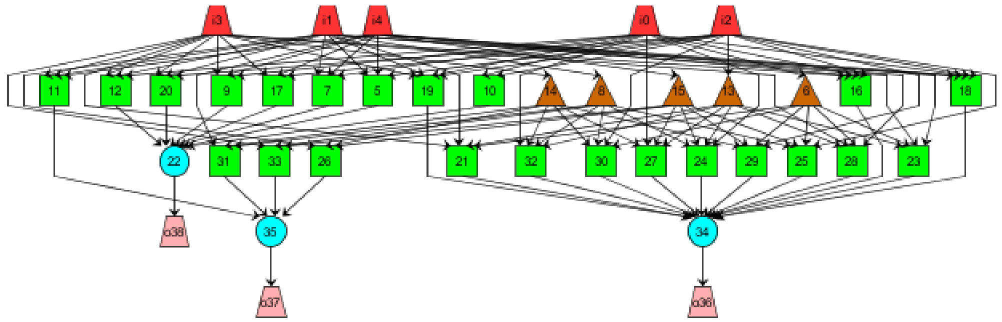
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

29=AND(1,3)
66=NOT(1)
25=AND(4,5)
. . .
. . .
. . .
48=NOT(47)
35=AND(28,34)
63=AND(28,32)
36=NOT(35)
64=NOT(63)
10=NOT(62)
49=AND(36,48)
65=AND(30,64)
16=NOT(49)
17=NOT(65)
```

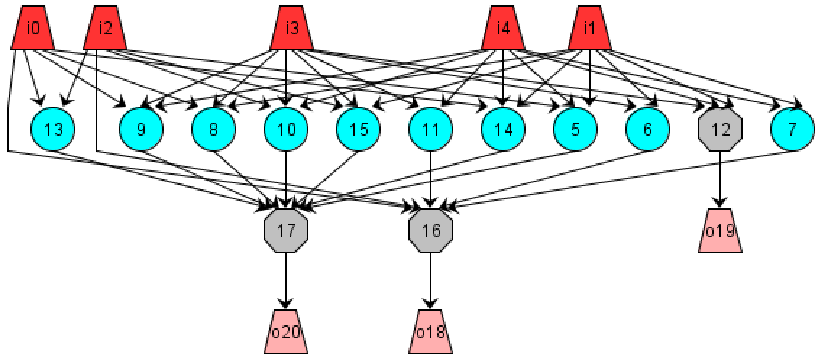
AIGs are not necessarily canonical

Transforms: Reduced Canonical Forms







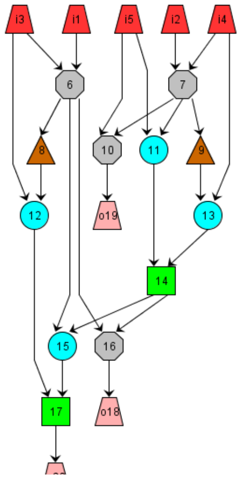


Sum-of-Minterms (reduced SOP)
Product-of-Maxterms (reduced POS)
ReedMuller (reduced RSE)

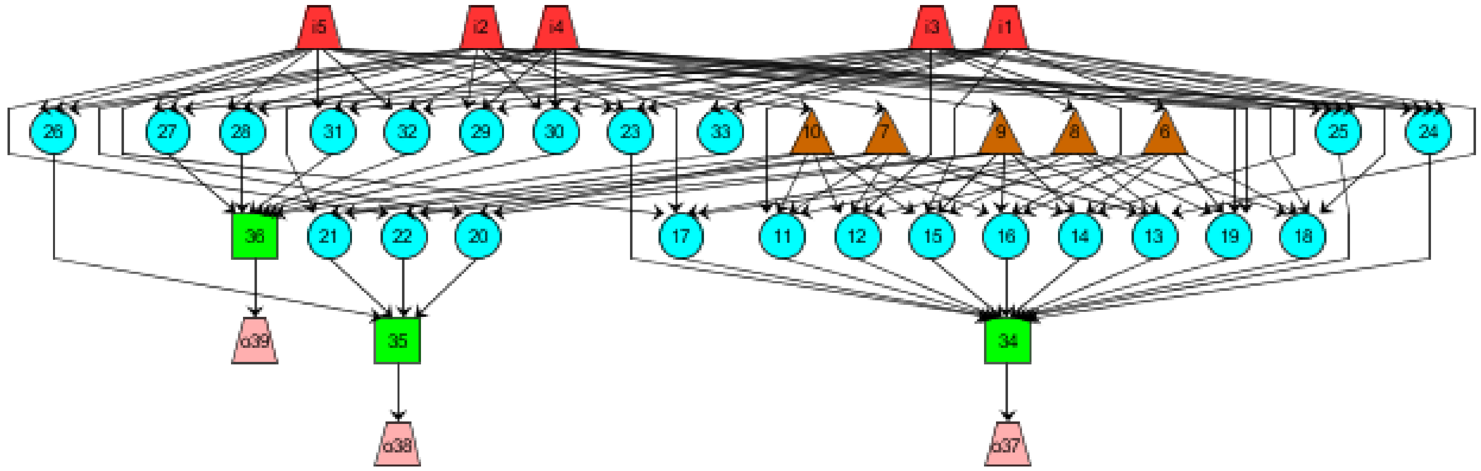
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Transforms: Espresso

Transforms a circuit based on its Espresso reduction (SOP factors) and equivalent BENCH



```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```

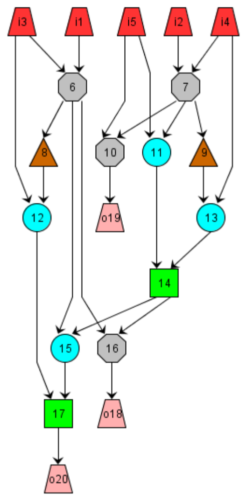


INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

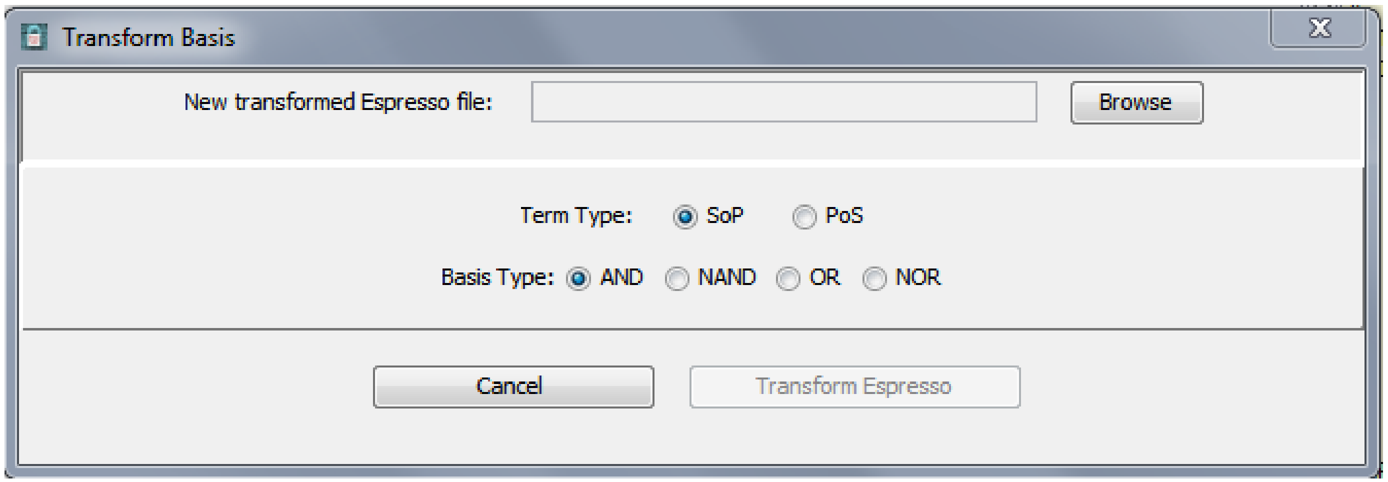
Transforms: Espresso Canonical Forms

Transforms a circuit based on its Espresso reduction and equivalent BENCH

There are 8 possible synthesis options based on SOP/POS and basis gate type



```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```



New transformed Espresso file:

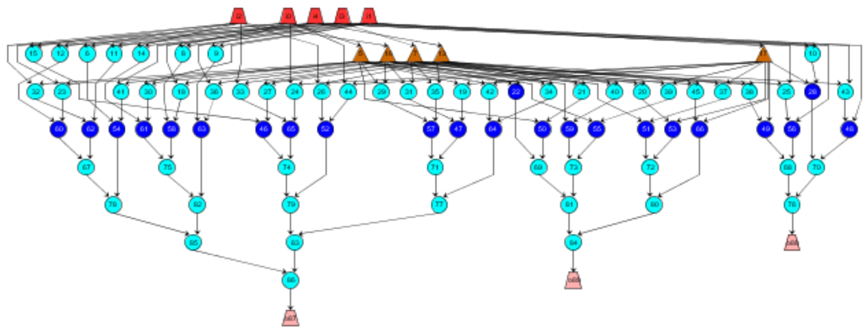
Term Type: [✓ SoP] PoS

Basis Type: [✓ AND] NAND OR NOR

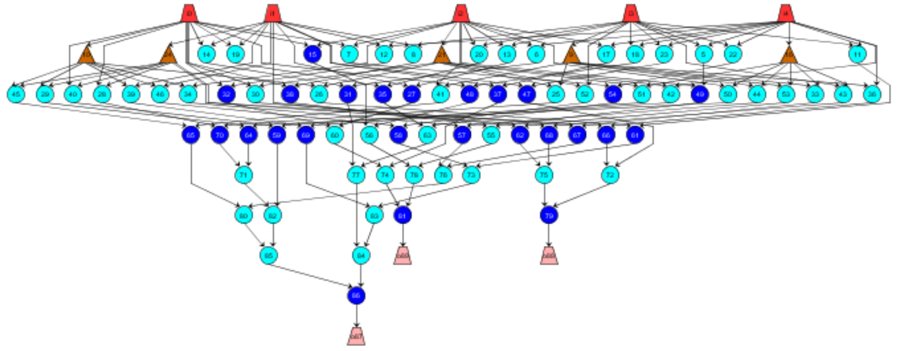
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Transforms: Espresso Canonical Forms

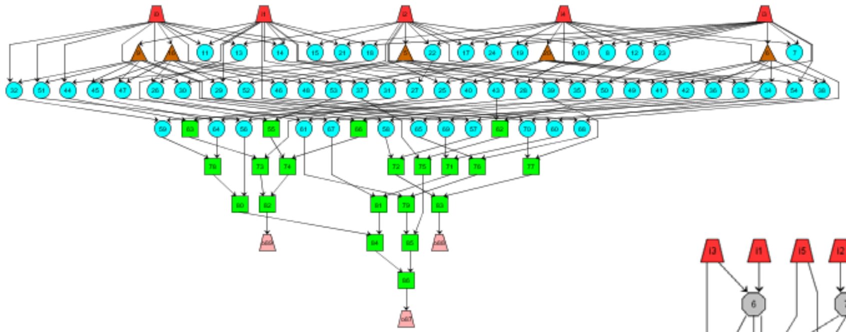
SOP-AND



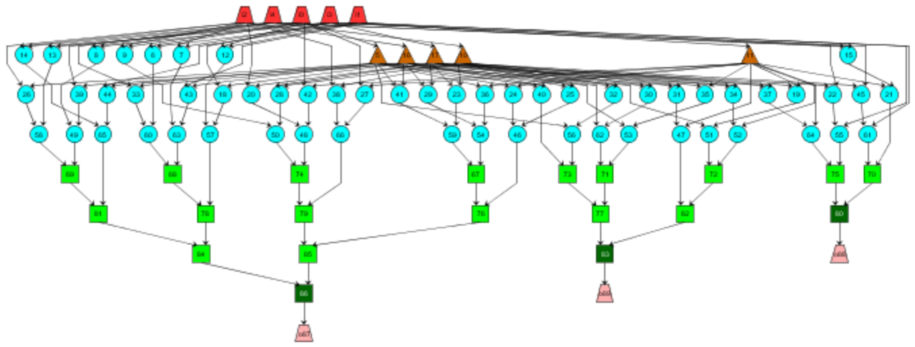
SOP-NAND



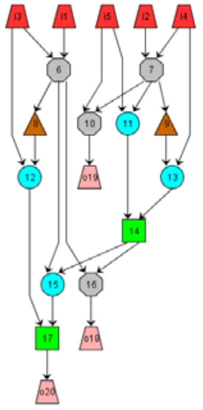
SOP-OR

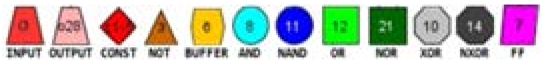


SOP-NOR



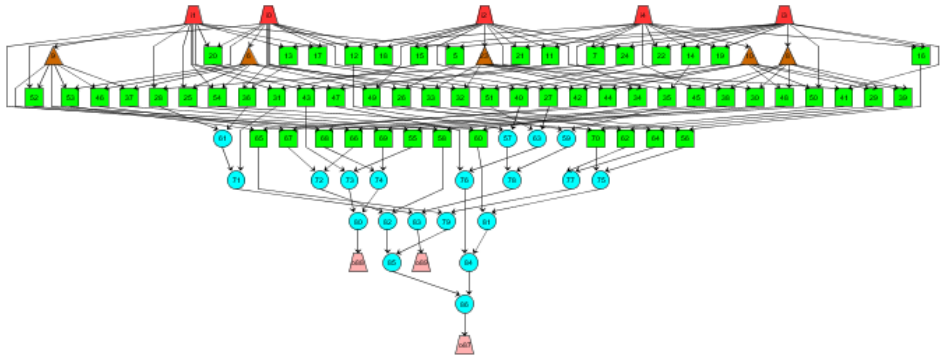
ORIGINAL



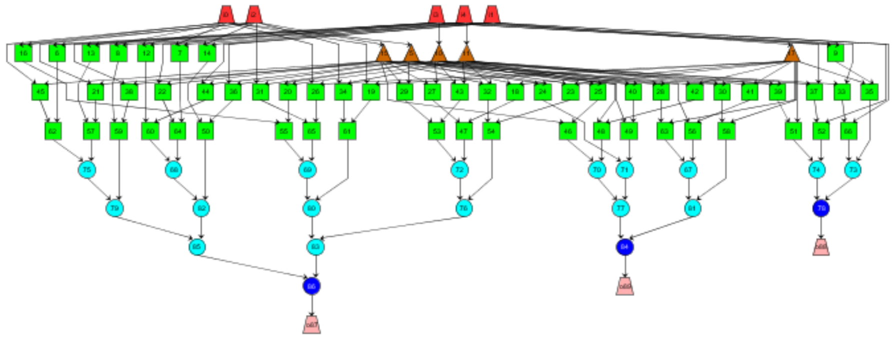


Transforms: Espresso Canonical Forms

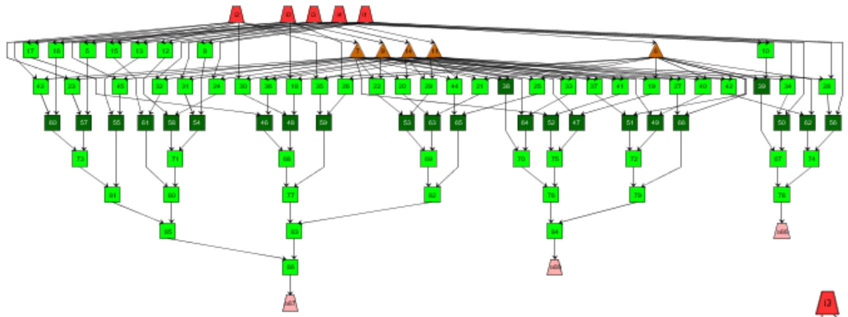
POS-AND



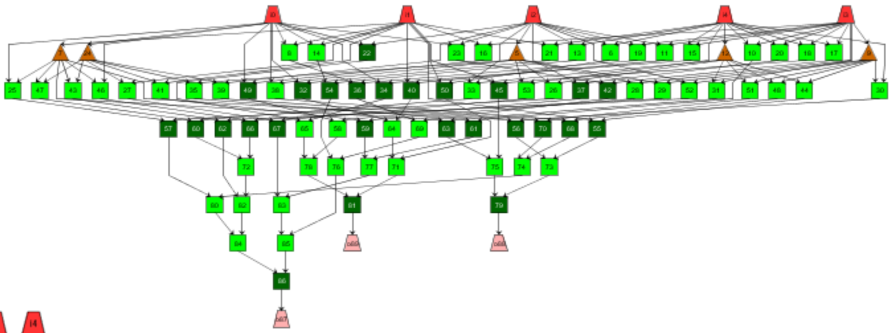
POS-NAND



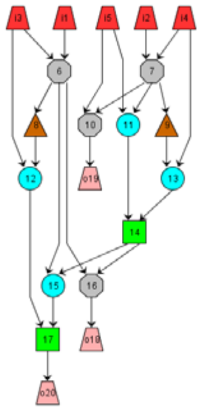
POS-OR

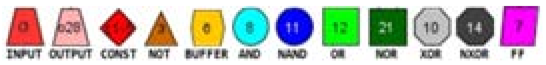


POS-NOR



ORIGINAL

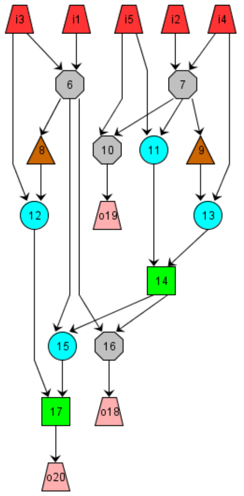




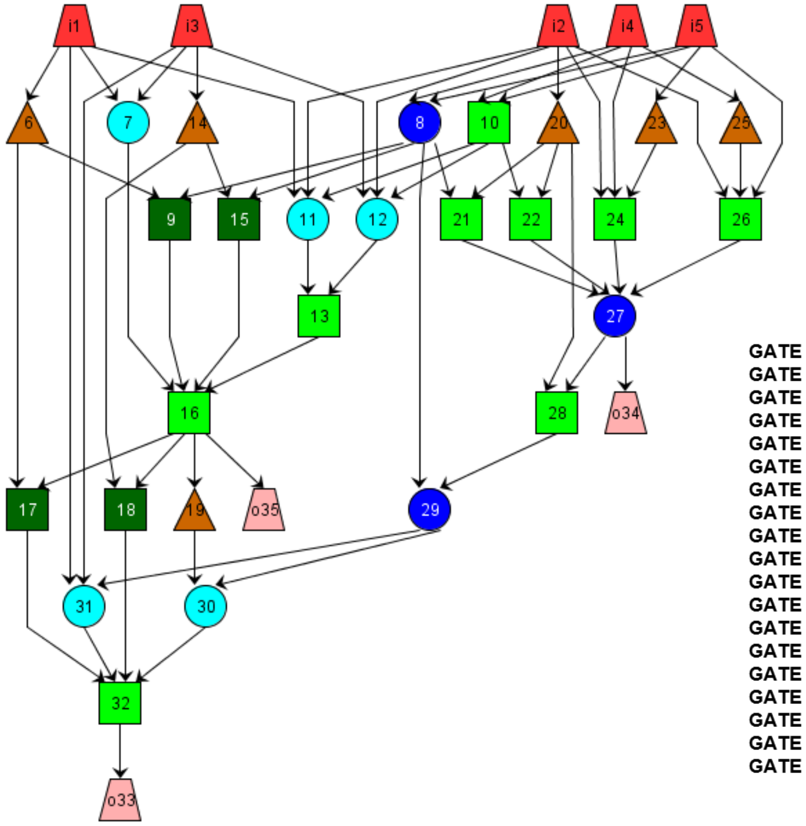
Transforms: misII

Transforms a circuit based on its misII reduction (SOP factors) and equivalent BENCH

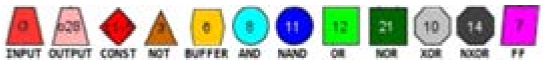
Gates are mapped according to the pet.genlib technology map



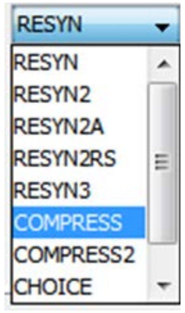
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(16)
OUTPUT(10)
OUTPUT(17)

6=XOR(1,3)
7=XOR(2,4)
9=NOT(7)
10=XOR(5,7)
11=AND(5,7)
8=NOT(6)
12=AND(3,8)
13=AND(4,9)
14=OR(11,13)
15=AND(14,6)
16=XOR(14,6)
17=OR(15,12)
```

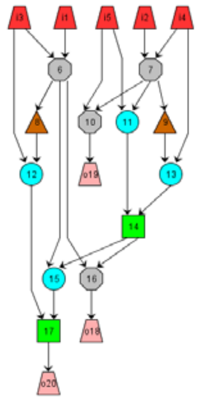


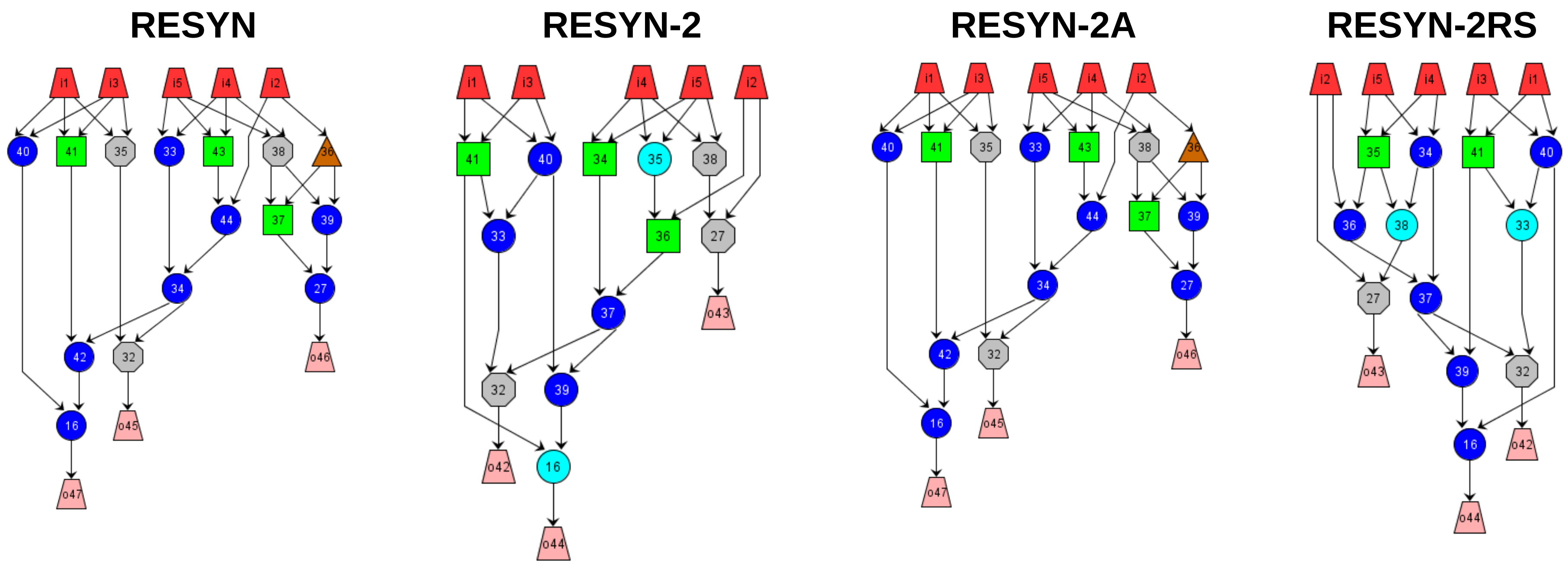
```
GATE zero0O=CONST0;
GATE one0O=CONST1;
GATE inv12O=!a;PIN * INV  1 999 0.9 0.3 0.9 0.3
GATE and23O=a*b;PIN * NONINV  1 999 1 0.2 1 0.2
GATE and34O=a*b*c;PIN * NONINV  1 999 1 0.2 1 0.2
GATE and45O=a*b*c*d;PIN * NONINV  1 999 1 0.2 1 0.2
GATE or23O=a+b;PIN * NONINV  1 999 1 0.2 1 0.2
GATE or34O=a+b+c;PIN * NONINV  1 999 1 0.2 1 0.2
GATE or45O=a+b+c+d;PIN * NONINV  1 999 1 0.2 1 0.2
GATE nand23O=!(a*b);PIN * INV  1 999 1.0 0.2 1.0 0.2
GATE nand34O=!(a*b*c);PIN * INV  1 999 1.1 0.3 1.1 0.3
GATE nand4 5O=!(a*b*c*d);PIN * INV  1 999 1.4 0.4 1.4 0.4
GATE nor23O=!(a+b);PIN * INV  1 999 1.4 0.5 1.4 0.5
GATE nor34O=!(a+b+c);PIN * INV  1 999 2.4 0.7 2.4 0.7
GATE nor45O=!(a+b+c+d);PIN * INV  1 999 3.8 1.0 3.8 1.0
GATE xor25O=a*!b+!a*b;PIN * UNKNOWN  2 999 1.9 0.5 1.9 0.5
GATE xor25O=!(a*b+!a*!b);PIN * UNKNOWN  2 999 1.9 0.5 1.9 0.5
GATE xnor25O=a*b+!a*!b;PIN * UNKNOWN  2 999 2.1 0.5 2.1 0.5
GATE xnor25O=!(!a*b+a*!b);PIN * UNKNOWN  2 999 2.1 0.5 2.1 0.5
```

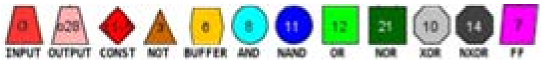


Transforms a circuit based on its ABC synthesis and equivalent BENCH
9 different synthesis scripts in ABC…



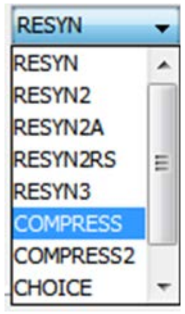


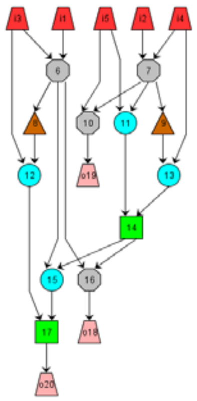




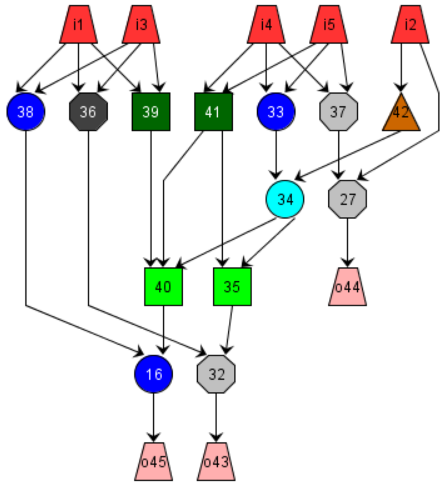
Transforms: ABC

Transforms a circuit based on its ABC synthesis and equivalent BENCH
9 different synthesis scripts in ABC…

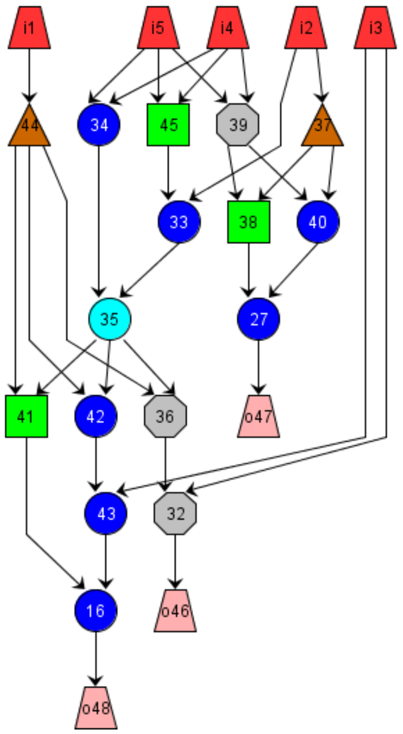




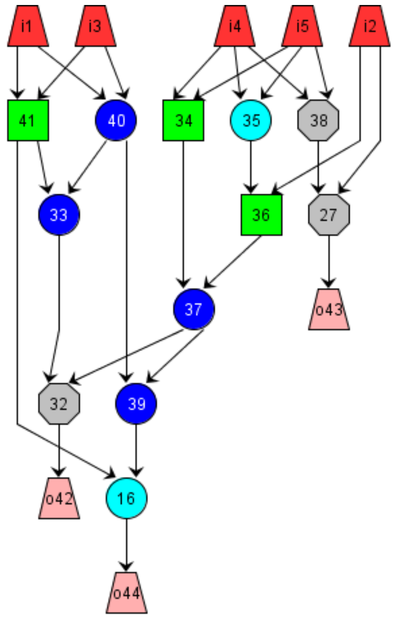
RESYN3



COMPRESS



COMPRESS2

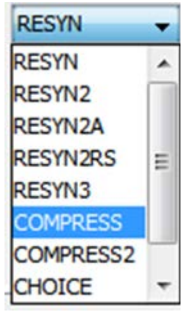


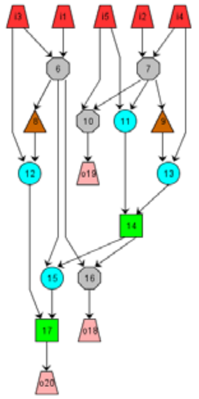
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Transforms: ABC

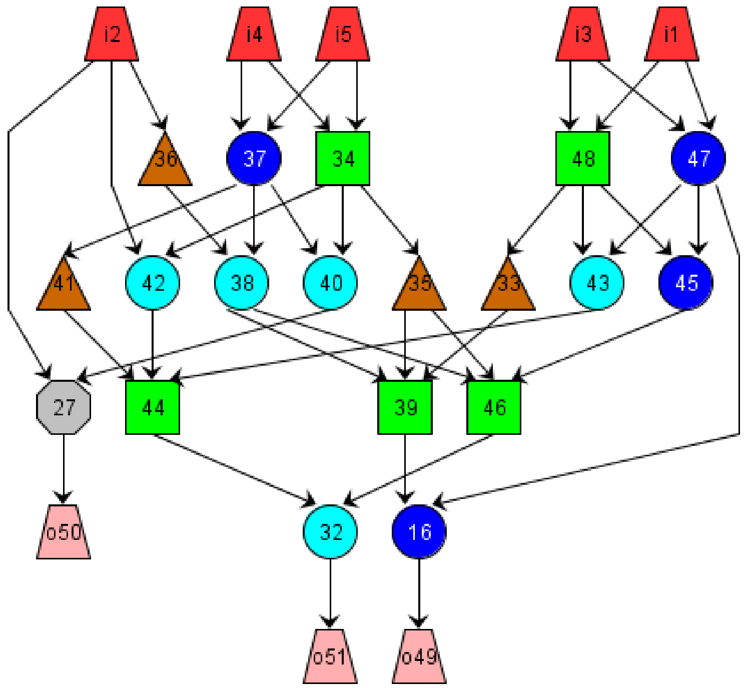
Transforms a circuit based on its ABC synthesis and equivalent BENCH

9 different synthesis scripts in ABC…

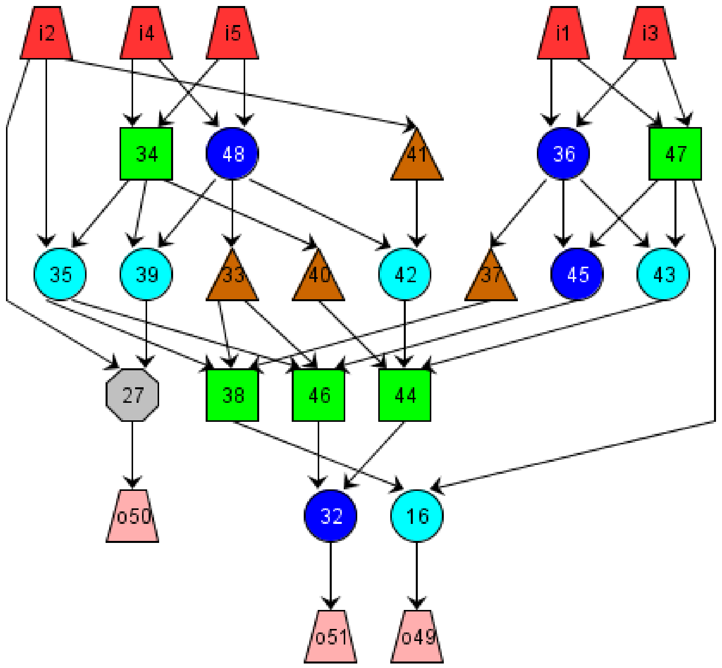


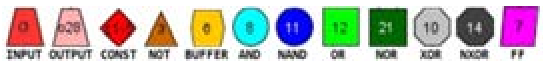


CHOICE



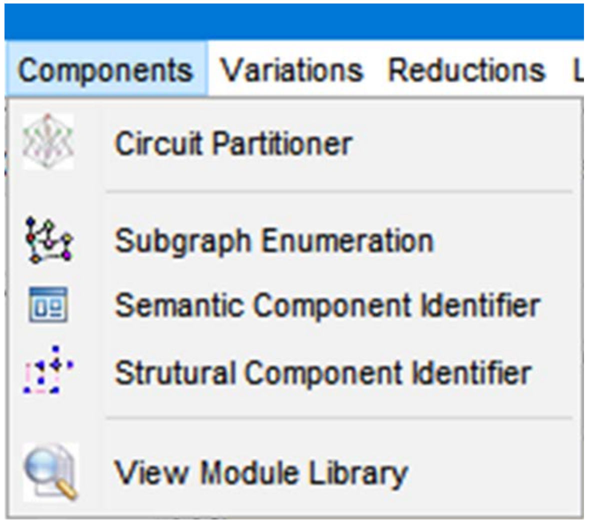
CHOICE2





Components:

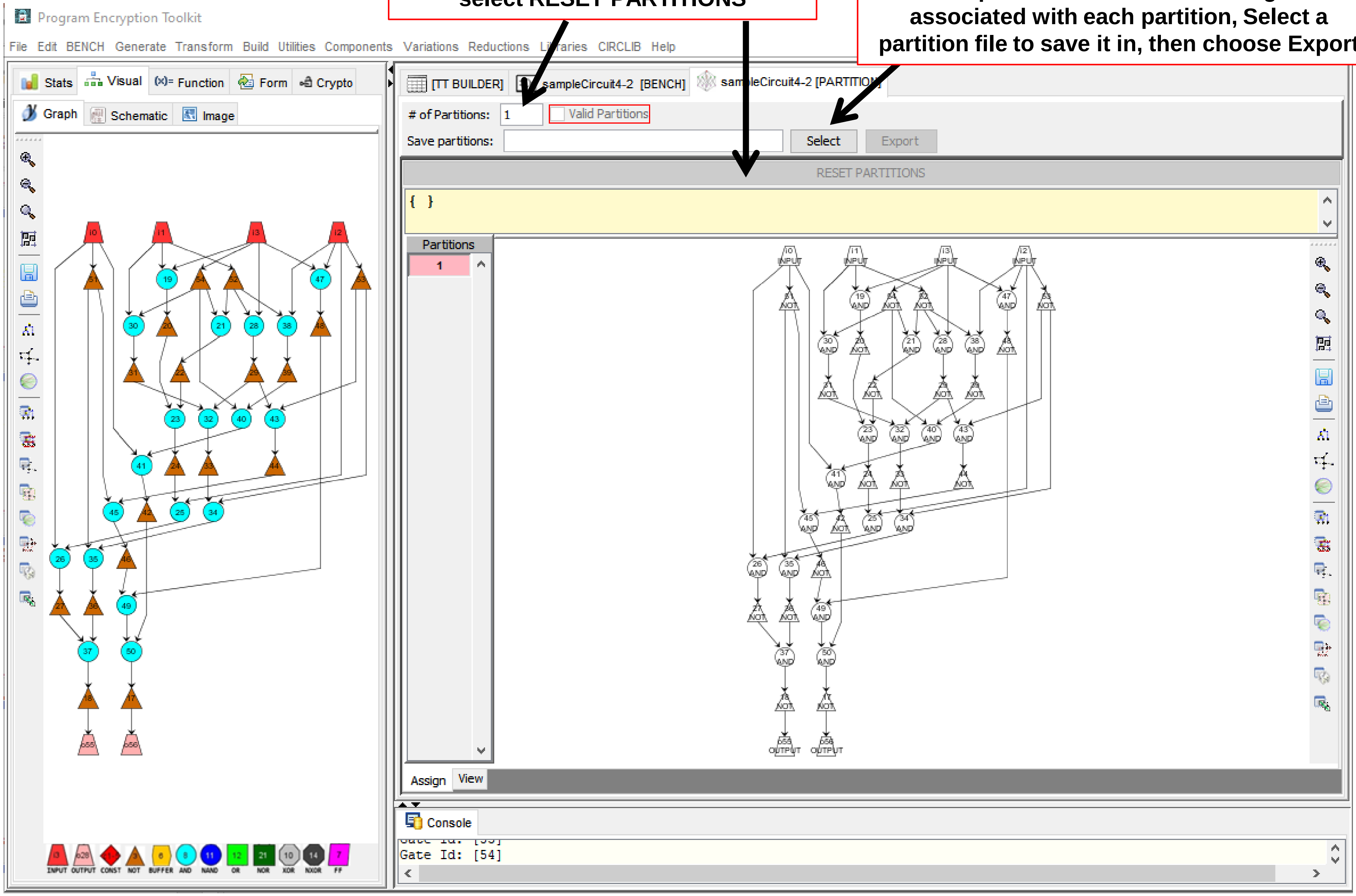
Circuit Partitioner
Subgraph Enumeration
Semantic Component Identification
Structural Component Identification
View Module Library



Circuit Partitioner

Pick the # of partitions first, then select RESET PARTITIONS

Once partitions are chosen with gates associated with each partition, Select a partition file to save it in, then choose Export

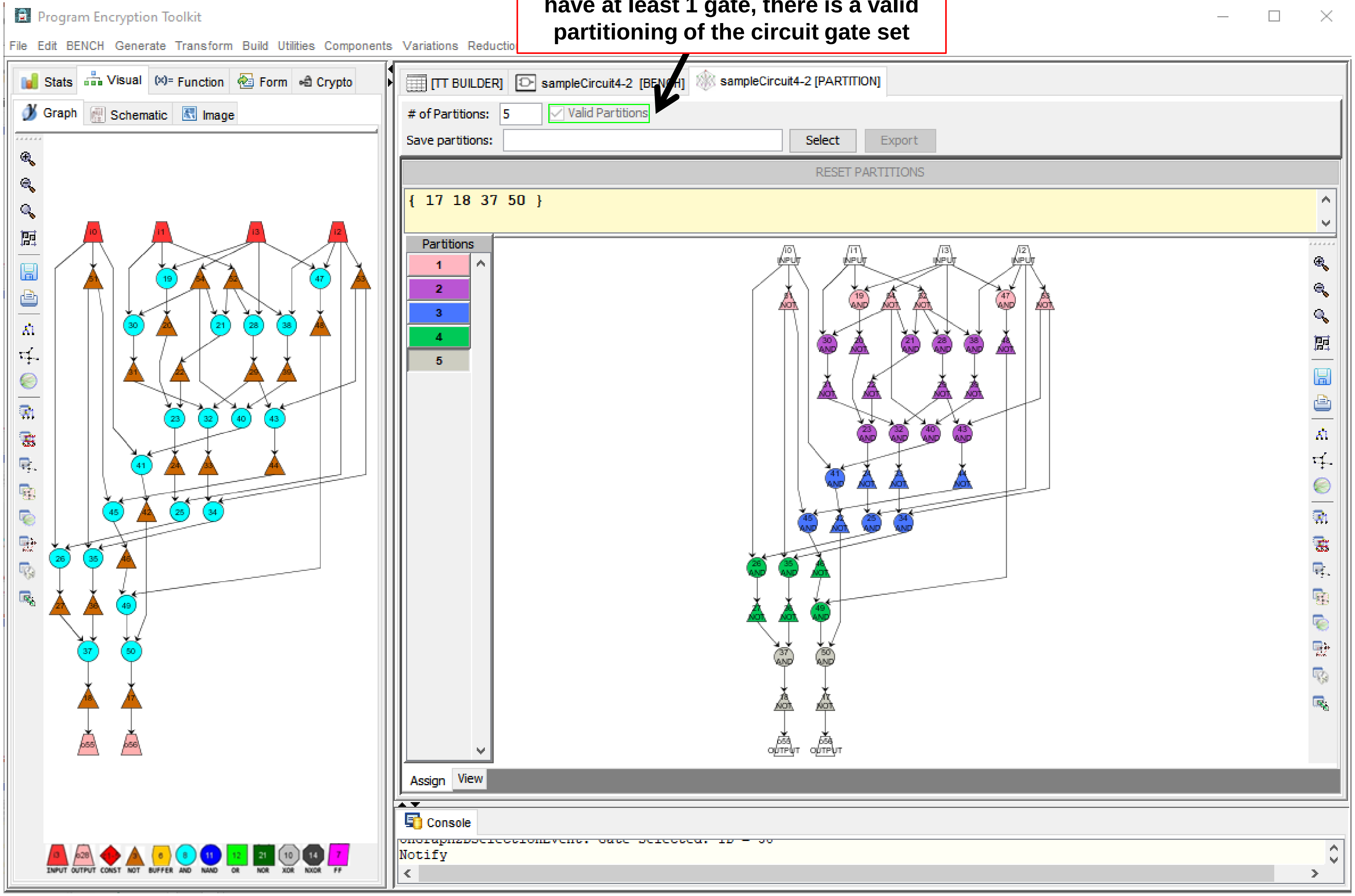

Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto
Graph Schematic Image
[TT BUILDER] sampleCircuit4-2 [BENCH] sampleCircuit4-2 [PARTITION]
# of Partitions: 1
Valid Partitions
Save partitions:
Select
Export
RESET PARTITIONS
Partitions
Assign View
Console
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF

```
{ }
```

```
Gate Id: [54]
```

Circuit Partitioner

Once all gates are assigned to at least 1 partition, and all partitions have at least 1 gate, there is a valid partitioning of the circuit gate set


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductio
Stats Visual Function Form Crypto
Graph Schematic Image
[TT BUILDER] sampleCircuit4-2 [BENCH] sampleCircuit4-2 [PARTITION]
RESET PARTITIONS
Partitions
1
2
3
4
5
Assign View
Console
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF

# of Partitions: 5
☑ Valid Partitions
Save partitions:
Select Export

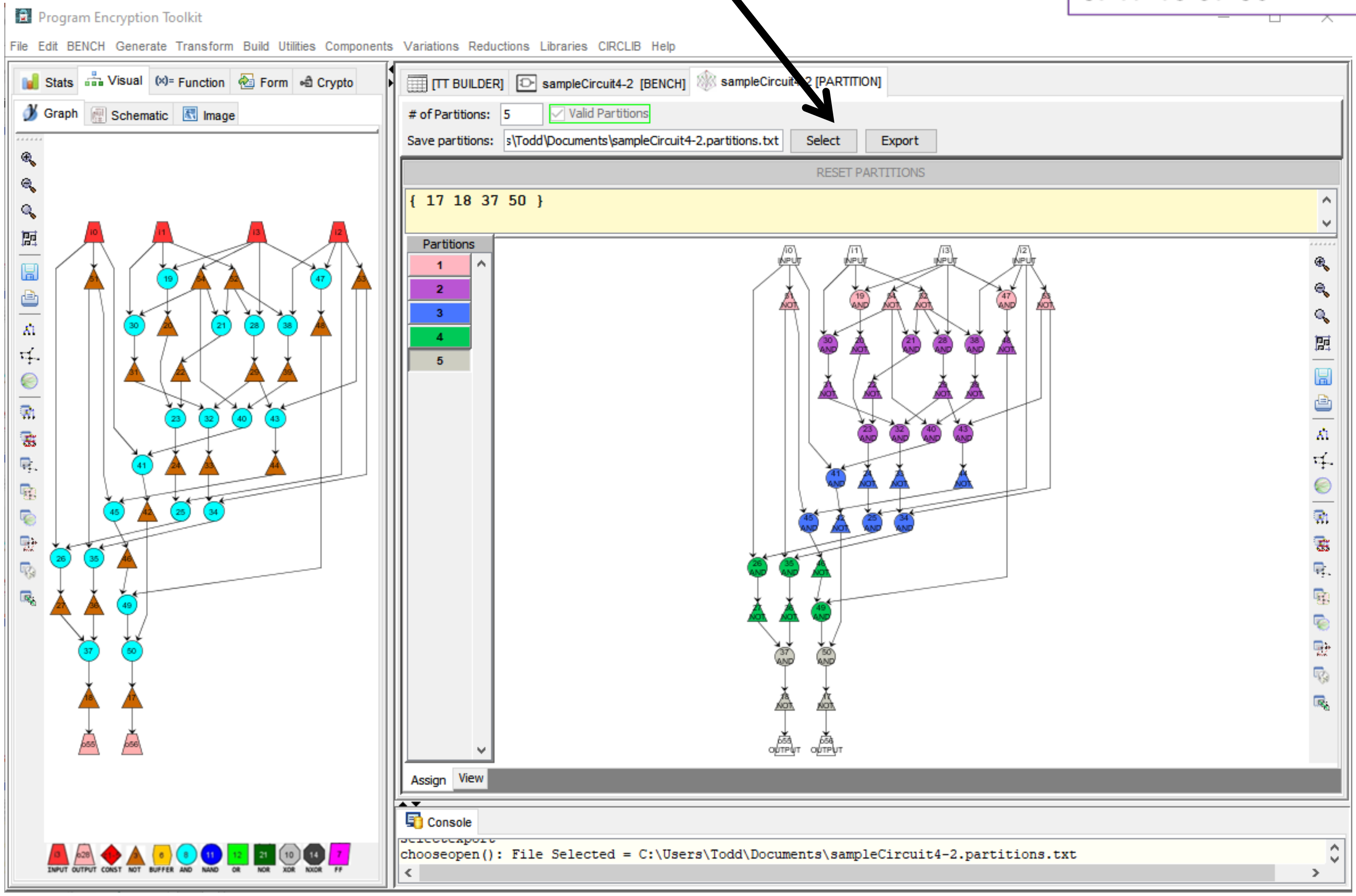
```
{ 17 18 37 50 }
```

```
Notify
```

Circuit Partitioner

With save file Selected and Exported, a text file will contain the partition information

```
1: 19 47 51 52 53 54
2: 20 21 22 23 28 29 30 31 32 38 39 40 43 48
3: 24 25 33 34 41 42 44 45
4: 26 27 35 36 46 49
5: 17 18 37 50
```

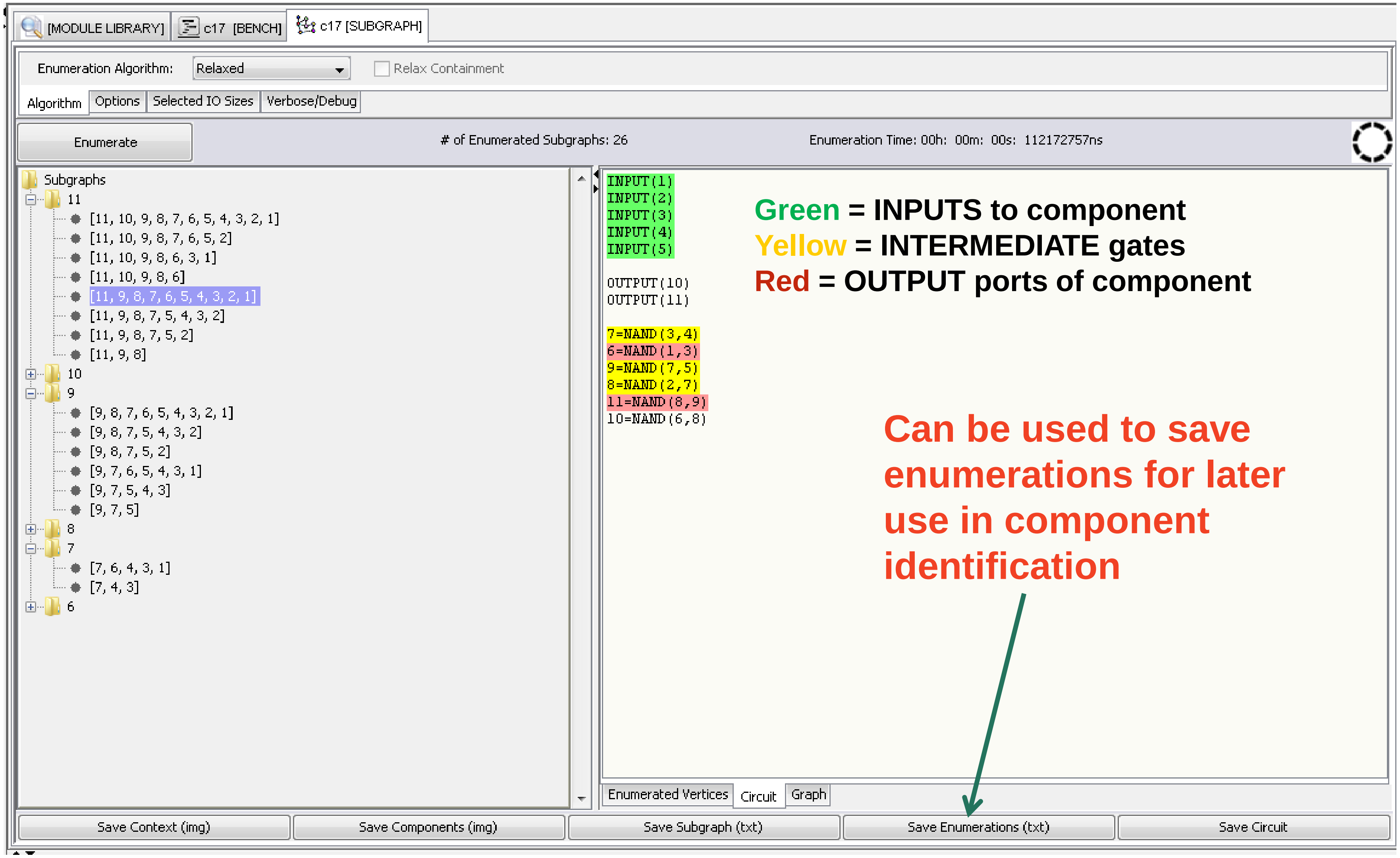

Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto
Graph Schematic Image
[TT BUILDER] sampleCircuit4-2 [BENCH] sampleCircuit4-2 [PARTITION]
Valid Partitions
Select
Export
RESET PARTITIONS
Partitions
Assign View
Console

# of Partitions: 5
Save partitions: s\Todd\Documents\sampleCircuit4-2.partitions.txt

```
{ 17 18 37 50 }
```

```
chooseopen(): File Selected = C:\Users\Todd\Documents\sampleCircuit4-2.partitions.txt
```

Subgraph Enumeration



Enumeration Algorithm: Relaxed

Relax Containment

# of Enumerated Subgraphs: 26

Enumeration Time: 00h: 00m: 00s: 112172757ns

```
Subgraphs
  11
    [11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    [11, 10, 9, 8, 7, 6, 5, 2]
    [11, 10, 9, 8, 6, 3, 1]
    [11, 10, 9, 8, 6]
    [11, 9, 8, 7, 6, 5, 4, 3, 2, 1]
    [11, 9, 8, 7, 5, 4, 3, 2]
    [11, 9, 8, 7, 5, 2]
    [11, 9, 8]
  10
  9
    [9, 8, 7, 6, 5, 4, 3, 2, 1]
    [9, 8, 7, 5, 4, 3, 2]
    [9, 8, 7, 5, 2]
    [9, 7, 6, 5, 4, 3, 1]
    [9, 7, 5, 4, 3]
    [9, 7, 5]
  8
  7
    [7, 6, 4, 3, 1]
    [7, 4, 3]
  6
```

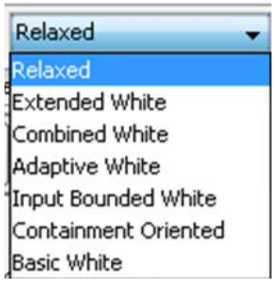
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(10)
OUTPUT(11)

7=NAND(3,4)
6=NAND(1,3)
9=NAND(7,5)
8=NAND(2,7)
11=NAND(8,9)
10=NAND(6,8)
```

Green = **INPUTS to component**
Yellow = **INTERMEDIATE gates**
Red = **OUTPUT ports of component**

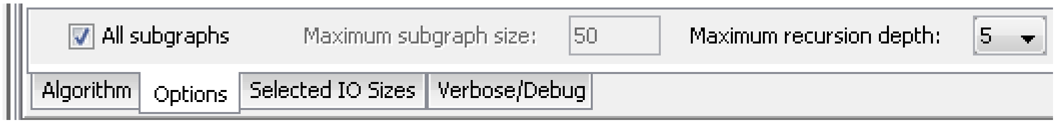
Can be used to save enumerations for later use in component identification

Subgraph Enumeration

- 7 different algorithms for enumeration=>
  - Relax containment: algorithms can required a component to be fully contained



- Generation options:

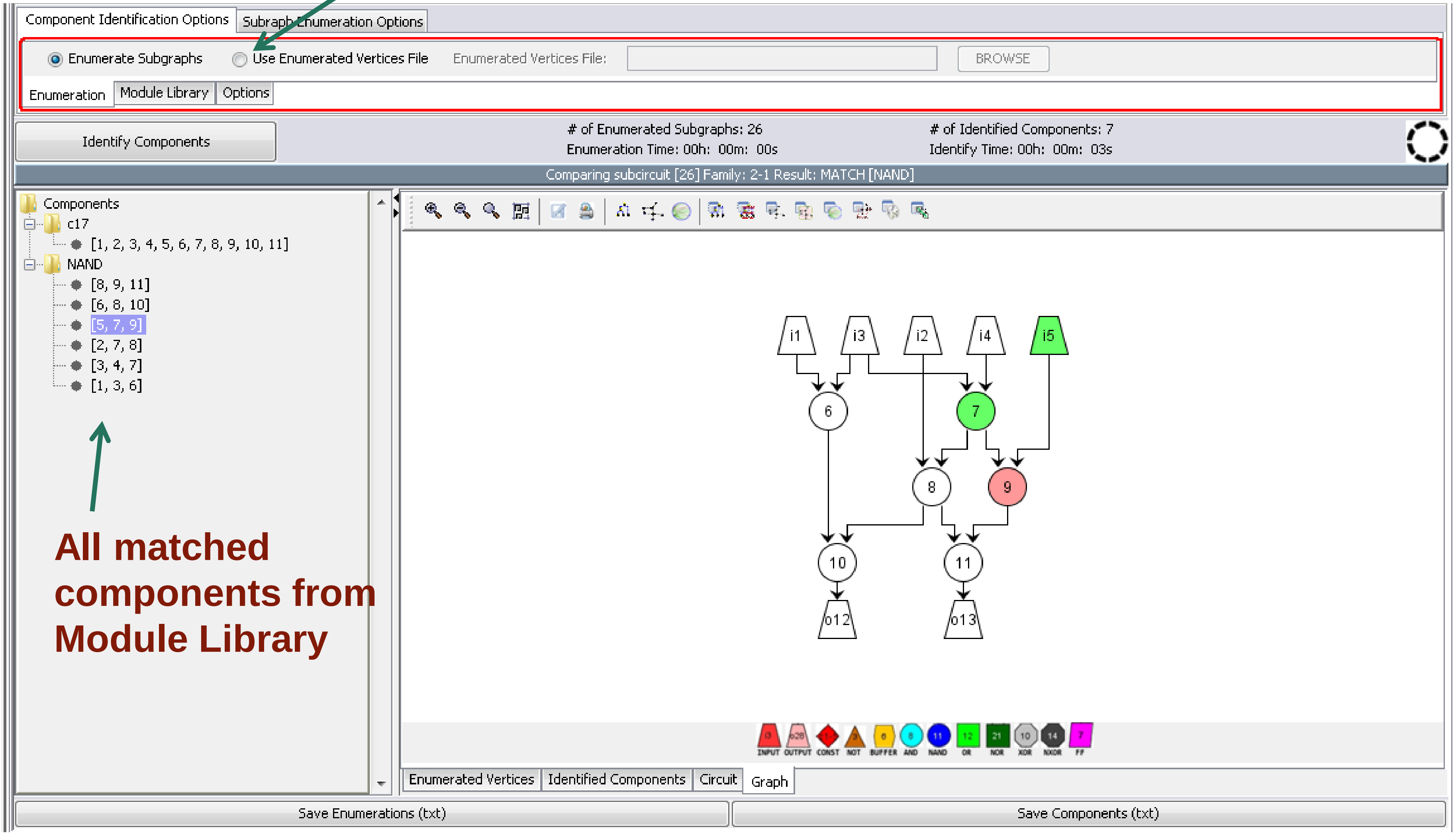

All subgraphs [✓]
Maximum subgraph size: 50
Maximum recursion depth: 5

  - All subgraphs OR limit enumeration to subgraphs of a given size (size includes inputs, outputs, and intermediate nodes of represented subcircuit)

  - Maximum recursion depth: for certain algorithms, used to control the amount of recursion for exploring subgraphs from a given starting node

Semantic Component Identification

Use a pre-generated enumeration file instead of enumerating from scratch



Component Identification Options | Subgraph Enumeration Options

[✓ Enumerate Subgraphs] Use Enumerated Vertices File Enumerated Vertices File: BROWSE

Enumeration | Module Library | Options

Identify Components

# of Enumerated Subgraphs: 26
Enumeration Time: 00h: 00m: 00s

# of Identified Components: 7
Identify Time: 00h: 00m: 03s

Comparing subcircuit [26] Family: 2-1 Result: MATCH [NAND]

```
Components
  c17
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
  NAND
    [8, 9, 11]
    [6, 8, 10]
    [5, 7, 9]
    [2, 7, 8]
    [3, 4, 7]
    [1, 3, 6]
```

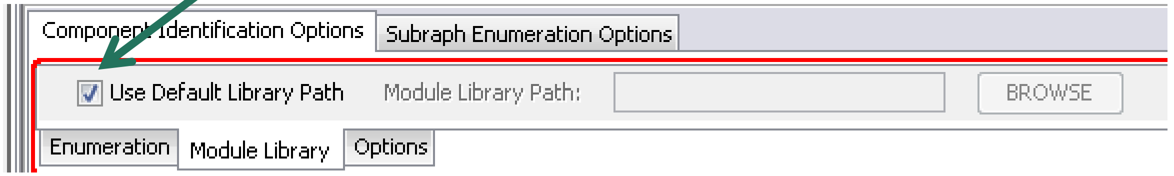
Enumerated Vertices | Identified Components | Circuit | Graph

Save Enumerations (txt) | Save Components (txt)

All matched components from Module Library

Semantic Component Identification

Module Library: use the default module library provided with PET or create your own version



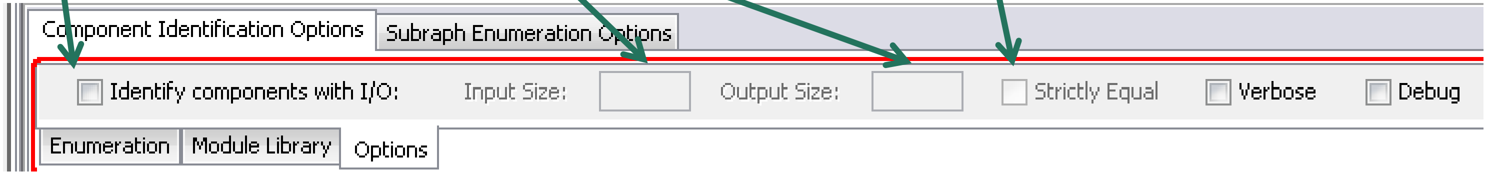
Component Identification Options | Subraph Enumeration Options

[✓] Use Default Library Path   Module Library Path: BROWSE

Enumeration | Module Library | Options

Limit the components that are matched by I/O size:

- **If checked: provide an INPUT/OUTPUT size**
- **All components with these sizes OR LESS will be compared**
- **Strictly equal: compare components with EXACTLY the INPUT/OUTPUT size specified**

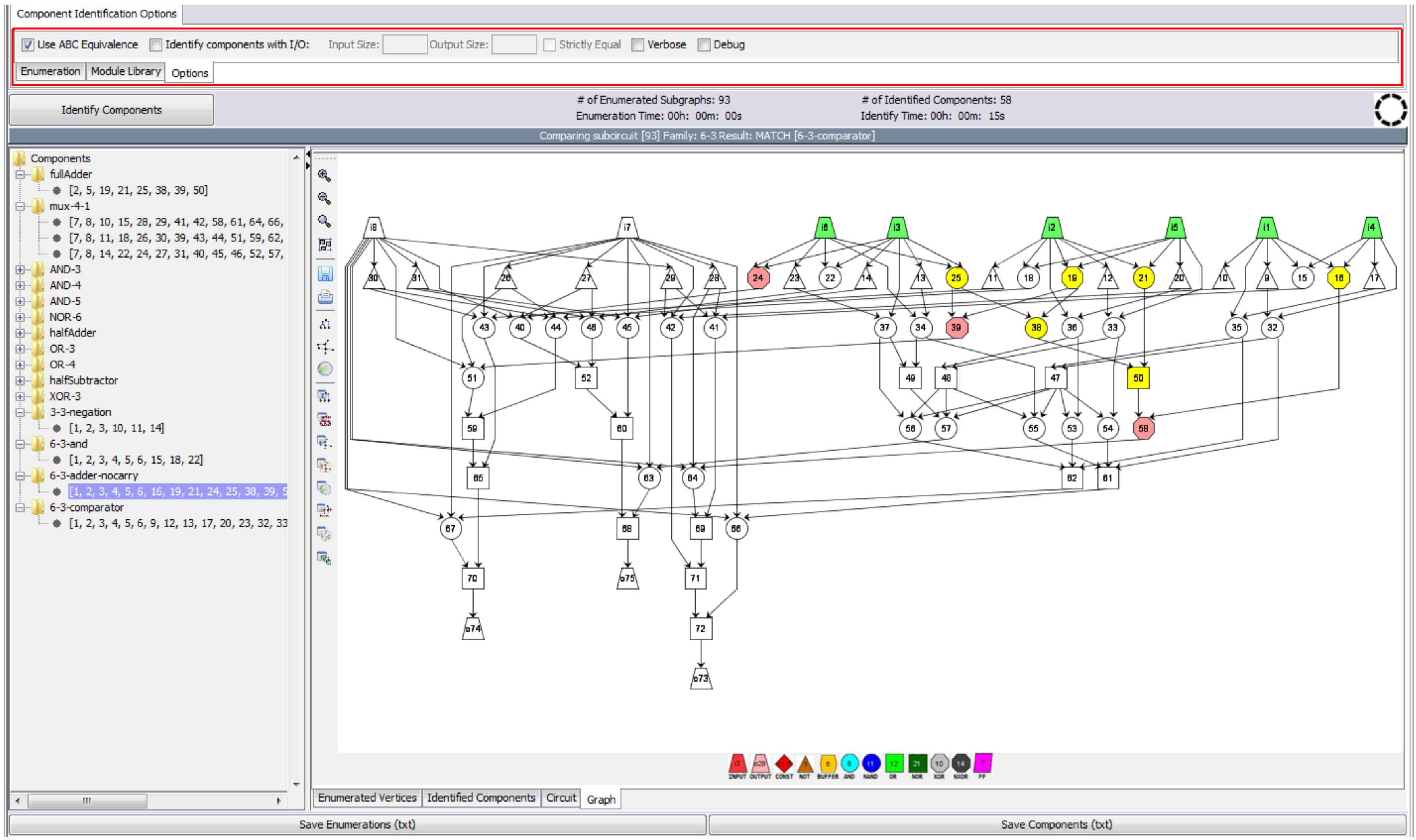


Component Identification Options | Subraph Enumeration Options

[ ] Identify components with I/O: Input Size: Output Size: [ ] Strictly Equal   [ ] Verbose   [ ] Debug

Enumeration | Module Library | Options

Semantic Component Identification



Component Identification Options

☑ Use ABC Equivalence ☐ Identify components with I/O: Input Size: Output Size: ☐ Strictly Equal ☐ Verbose ☐ Debug

Enumeration | Module Library | Options

Identify Components

# of Enumerated Subgraphs: 93
Enumeration Time: 00h: 00m: 00s

# of Identified Components: 58
Identify Time: 00h: 00m: 15s

Comparing subcircuit [93] Family: 6-3 Result: MATCH [6-3-comparator]

- Components
  - fullAdder
    - [2, 5, 19, 21, 25, 38, 39, 50]
  - mux-4-1
    - [7, 8, 10, 15, 28, 29, 41, 42, 58, 61, 64, 66,
    - [7, 8, 11, 18, 26, 30, 39, 43, 44, 51, 59, 62,
    - [7, 8, 14, 22, 24, 27, 31, 40, 45, 46, 52, 57,
  - AND-3
  - AND-4
  - AND-5
  - NOR-6
  - halfAdder
  - OR-3
  - OR-4
  - halfSubtractor
  - XOR-3
  - 3-3-negation
    - [1, 2, 3, 10, 11, 14]
  - 6-3-and
    - [1, 2, 3, 4, 5, 6, 15, 18, 22]
  - 6-3-adder-nocarry
    - [1, 2, 3, 4, 5, 6, 16, 19, 21, 24, 25, 38, 39, 5
  - 6-3-comparator
    - [1, 2, 3, 4, 5, 6, 9, 12, 13, 17, 20, 23, 32, 33

Enumerated Vertices | Identified Components | Circuit | Graph

Save Enumerations (txt) | Save Components (txt)

Constraints

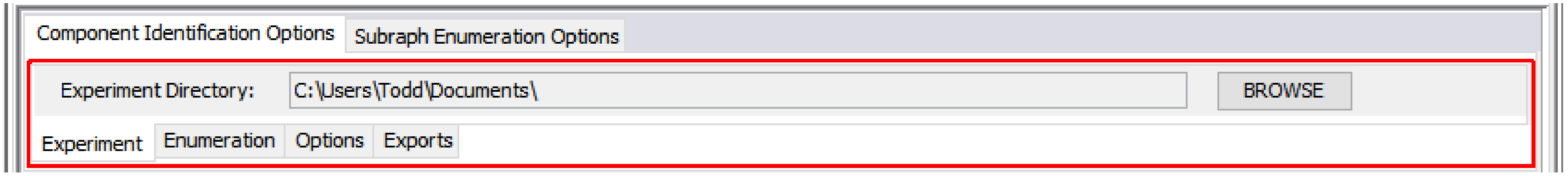
- Subgraph enumeration is N!, where N is size of circuit
  - So… ALL enumeration algorithms return approximations of total subgraphs that have the best potential to be a valid subcircuit component
  - The LARGER the starting circuit, the LARGER the # of subgraphs enumerated: memory and time tradeoffs begin to occur around 100K subcircuits

- Component identification is constrained by time and the # input/output size of the components being compared from the Module Library
  - Each enumerated subgraph is compared against a component from the library with a matching input size and output size
  - The match process generates all possible combinations of input ordering with all possible combinations of output orderings
  - This results in an X! * Y! number of combinations, where X is input size and Y is output size of the component being compared against
  - Therefore, components with about 6 or 7 inputs will take longer for any given comparison

- Based on the current implementations, it may be likely you will run into Java heap space or GC overhead limit exceptions
- Even with adequate RAM and specification of JVM options to utilize the space, time becomes the limiting factor for experiments

Structural Component Identifier

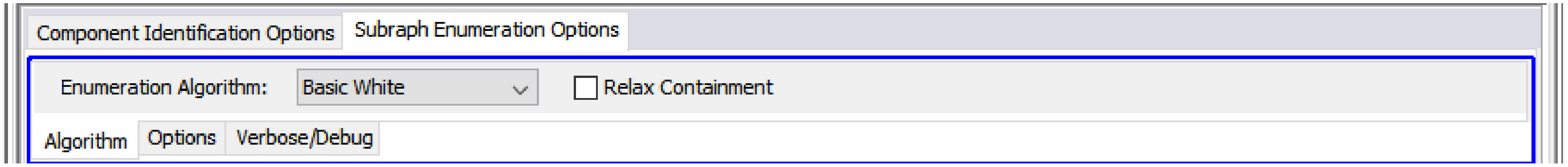
- Structural identification is much like semantic identification in terms of options:



Component Identification Options | Subraph Enumeration Options

Experiment Directory: C:\Users\Todd\Documents\ BROWSE

Experiment | Enumeration | Options | Exports



Component Identification Options | Subraph Enumeration Options

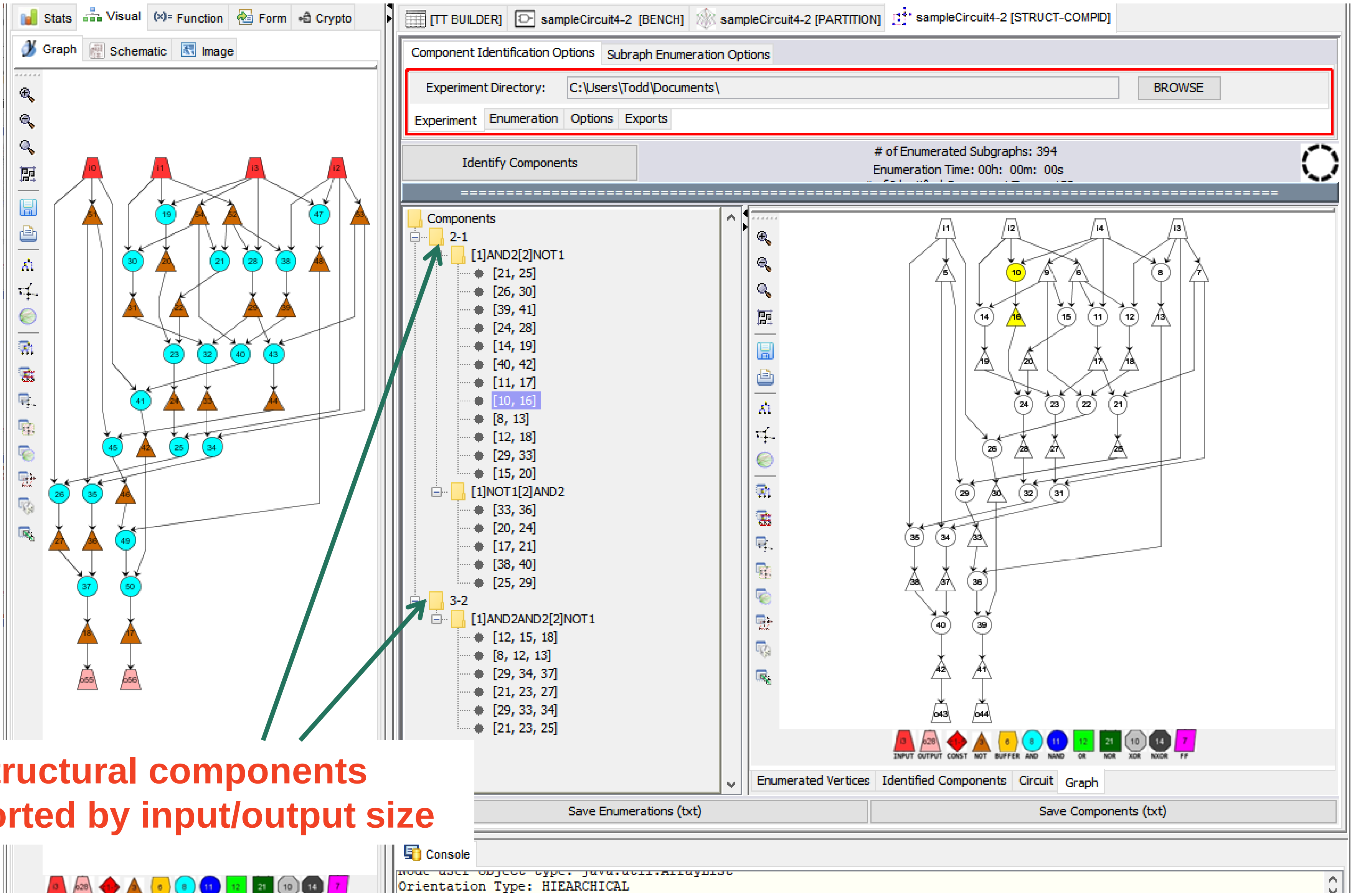
Enumeration Algorithm: Basic White ☐ Relax Containment

Algorithm | Options | Verbose/Debug

- Subgraphs are enumerated using a standard enumeration algorithm (1 of 7 must be selected)

- Identification involves finding common structures (not tied to any specific known component)

Structural Component Identifier



Experiment Directory: C:\Users\Todd\Documents\

# of Enumerated Subgraphs: 394
Enumeration Time: 00h: 00m: 00s

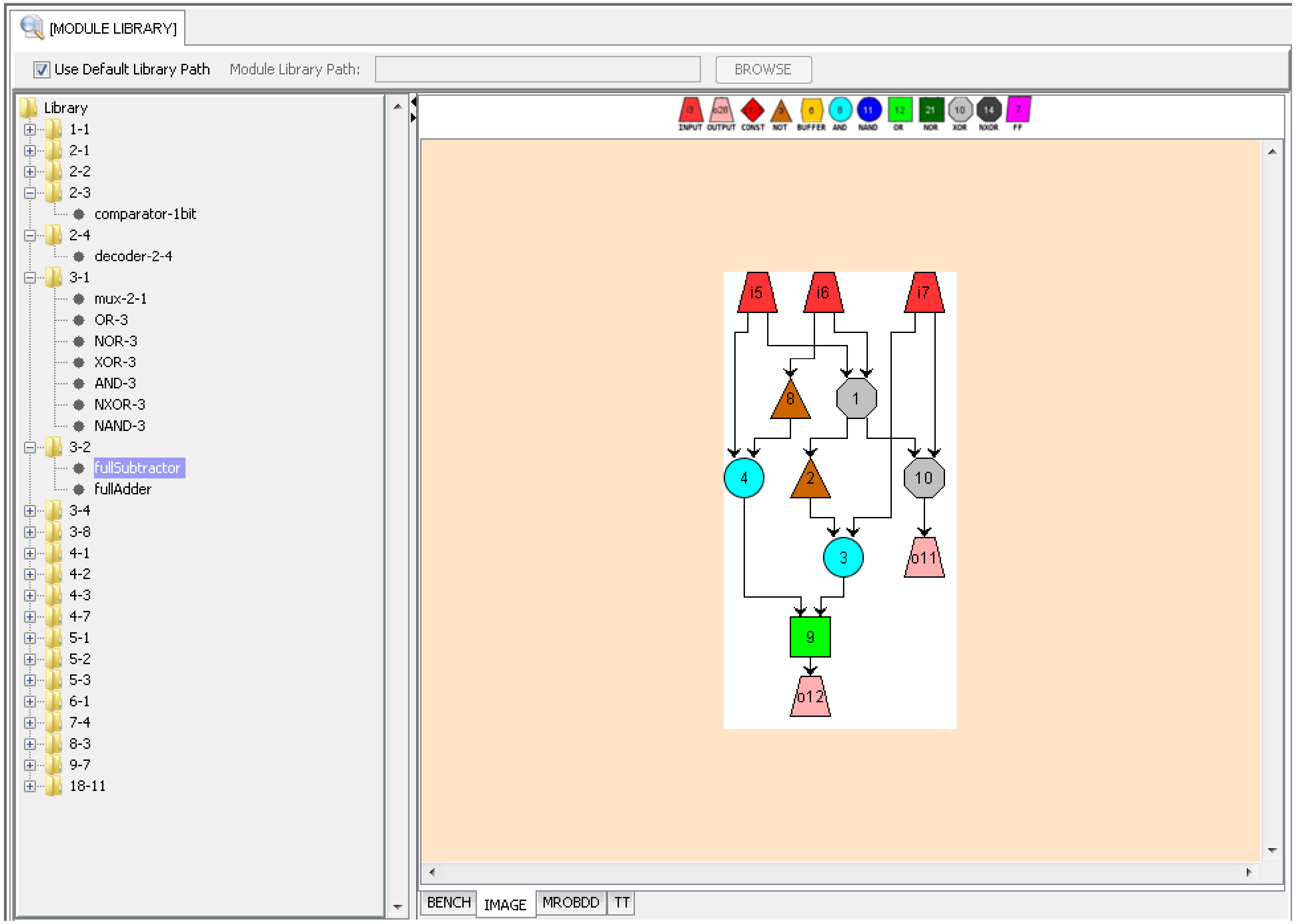
Components
- 2-1
  - [1]AND2[2]NOT1
    - [21, 25]
    - [26, 30]
    - [39, 41]
    - [24, 28]
    - [14, 19]
    - [40, 42]
    - [11, 17]
    - [10, 16]
    - [8, 13]
    - [12, 18]
    - [29, 33]
    - [15, 20]
  - [1]NOT1[2]AND2
    - [33, 36]
    - [20, 24]
    - [17, 21]
    - [38, 40]
    - [25, 29]
- 3-2
  - [1]AND2AND2[2]NOT1
    - [12, 15, 18]
    - [8, 12, 13]
    - [29, 34, 37]
    - [21, 23, 27]
    - [29, 33, 34]
    - [21, 23, 25]

```
Orientation Type: HIEARCHICAL
```

Structural components sorted by input/output size

View Module Library

Module library is specially formatted directory with components used in components identification experiments: components are arranged by input/output size and are defined as BENCH circuits



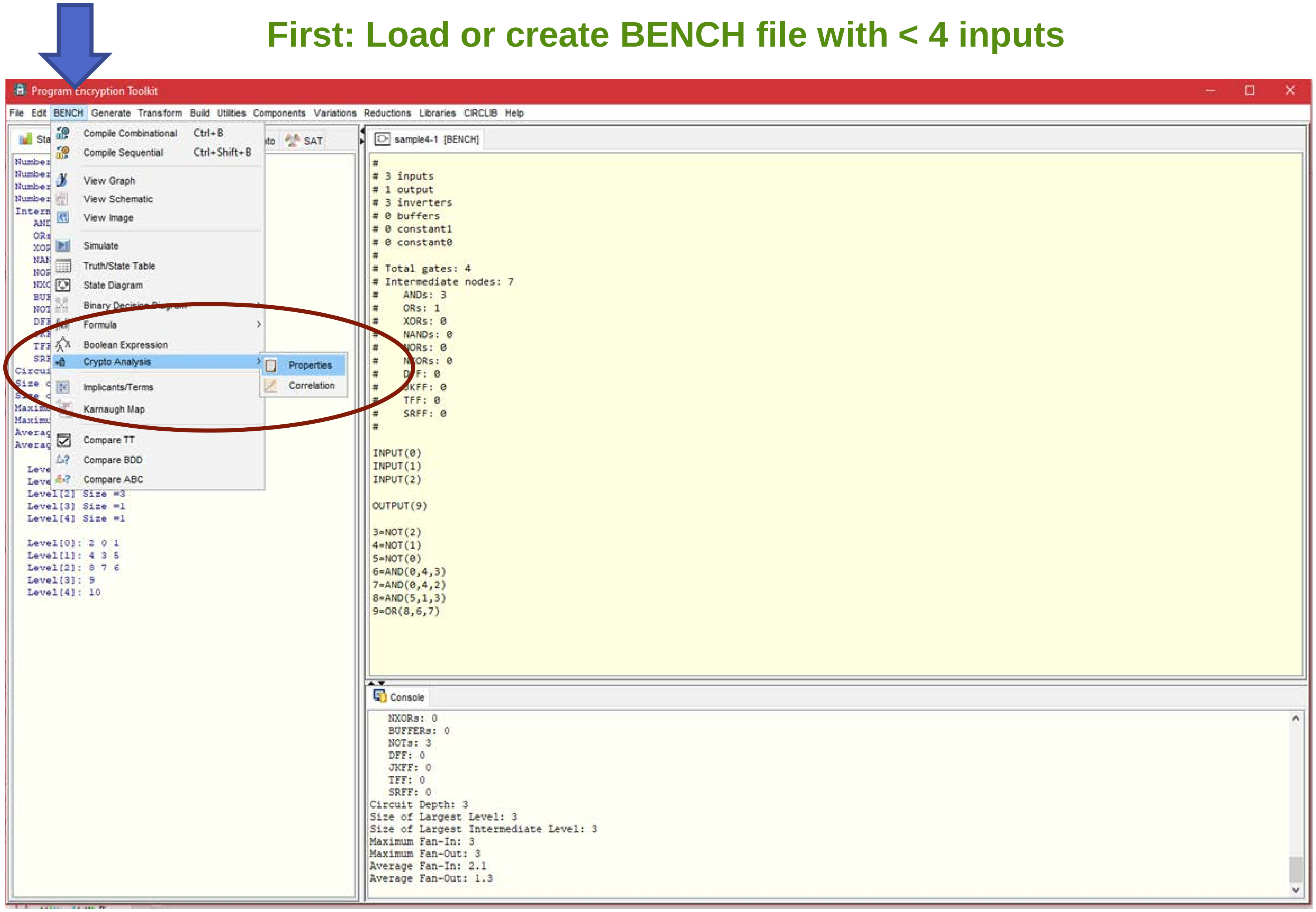
[MODULE LIBRARY]

☑ Use Default Library Path   Module Library Path: ____   BROWSE

- Library
  - 1-1
  - 2-1
  - 2-2
  - 2-3
    - comparator-1bit
  - 2-4
    - decoder-2-4
  - 3-1
    - mux-2-1
    - OR-3
    - NOR-3
    - XOR-3
    - AND-3
    - NXOR-3
    - NAND-3
  - 3-2
    - fullSubtractor
    - fullAdder
  - 3-4
  - 3-8
  - 4-1
  - 4-2
  - 4-3
  - 4-7
  - 5-1
  - 5-2
  - 5-3
  - 6-1
  - 7-4
  - 8-3
  - 9-7
  - 18-11

Cryptographic Boolean Properties

First: Load or create BENCH file with < 4 inputs



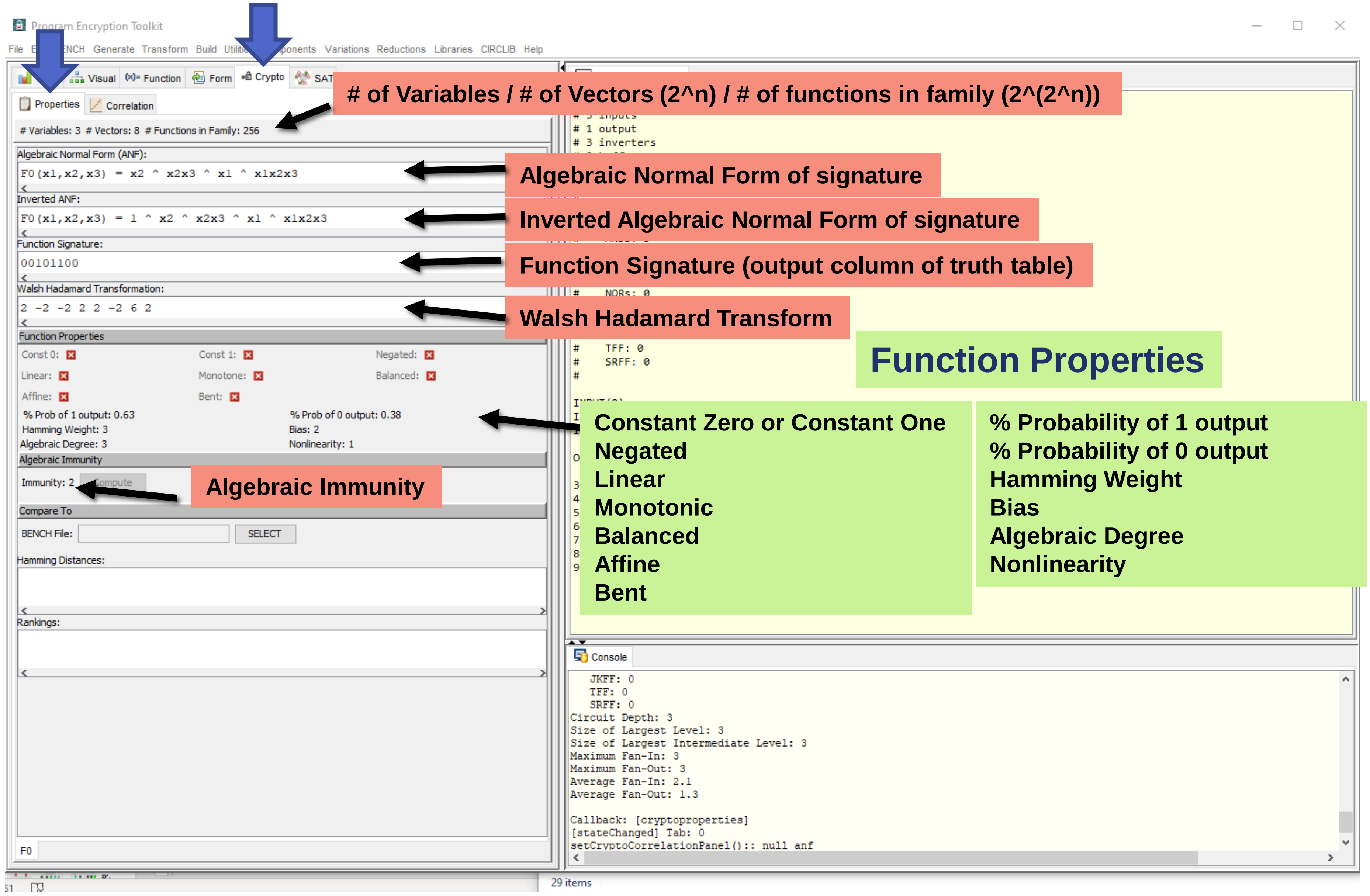
```
Level[2] Size =3
Level[3] Size =1
Level[4] Size =1

Level[0]: 2 0 1
Level[1]: 4 3 5
Level[2]: 8 7 6
Level[3]: 9
Level[4]: 10
```

```
#
# 3 inputs
# 1 output
# 3 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 4
# Intermediate nodes: 7
#    ANDs: 3
#    ORs: 1
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)

OUTPUT(9)

3=NOT(2)
4=NOT(1)
5=NOT(0)
6=AND(0,4,3)
7=AND(0,4,2)
8=AND(5,1,3)
9=OR(8,6,7)
```

```
   NXORs: 0
   BUFFERs: 0
   NOTs: 3
   DFF: 0
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 3
Size of Largest Level: 3
Size of Largest Intermediate Level: 3
Maximum Fan-In: 3
Maximum Fan-Out: 3
Average Fan-In: 2.1
Average Fan-Out: 1.3
```

Cryptographic Boolean Properties

BENCH->Crypto Analysis->Properties from main menu


Program Encryption Toolkit
File BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Visual Function Form Crypto SAT
Properties Correlation
# of Variables / # of Vectors (2^n) / # of functions in family (2^(2^n))
Algebraic Normal Form of signature
Inverted Algebraic Normal Form of signature
Function Signature (output column of truth table)
Walsh Hadamard Transform
Function Properties
Algebraic Immunity
Compute
SELECT
Console
29 items

Variables: 3 # Vectors: 8 # Functions in Family: 256

Algebraic Normal Form (ANF):

```
F0(x1,x2,x3) = x2 ^ x2x3 ^ x1 ^ x1x2x3
```

Inverted ANF:

```
F0(x1,x2,x3) = 1 ^ x2 ^ x2x3 ^ x1 ^ x1x2x3
```

Function Signature:

```
00101100
```

Walsh Hadamard Transformation:

```
2 -2 -2 2 2 -2 6 2
```

Function Properties

Const 0: ☒	Const 1: ☒	Negated: ☒
Linear: ☒	Monotone: ☒	Balanced: ☒
Affine: ☒	Bent: ☒	

% Prob of 1 output: 0.63	% Prob of 0 output: 0.38
Hamming Weight: 3	Bias: 2
Algebraic Degree: 3	Nonlinearity: 1

Algebraic Immunity

Immunity: 2

Compare To

BENCH File:

Hamming Distances:

Rankings:

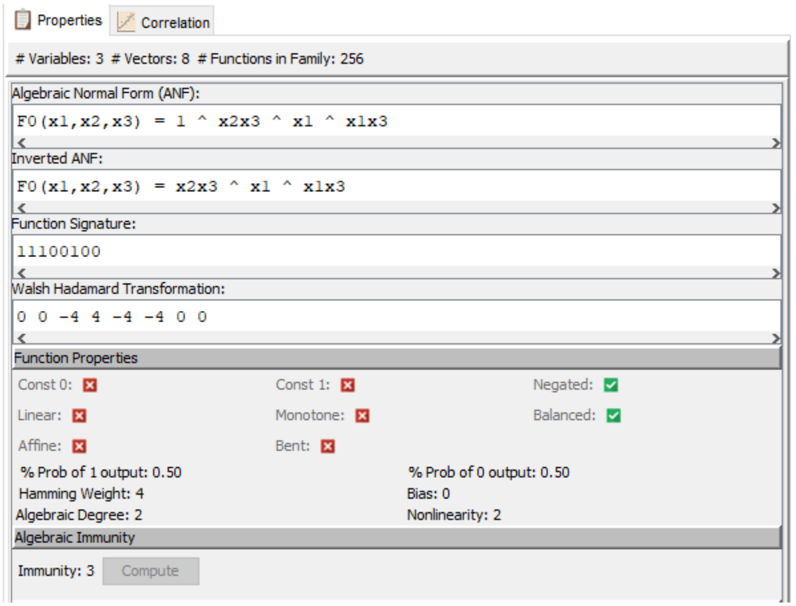
F0

```
# 3 inputs
# 1 output
# 3 inverters
#     NORs: 0
#     TFF: 0
#     SRFF: 0
#
```

```
   JKFF: 0
   TFF: 0
   SRFF: 0
Circuit Depth: 3
Size of Largest Level: 3
Size of Largest Intermediate Level: 3
Maximum Fan-In: 3
Maximum Fan-Out: 3
Average Fan-In: 2.1
Average Fan-Out: 1.3

Callback: [cryptoproperties]
[stateChanged] Tab: 0
setCryptoCorrelationPanel():: null anf
```

Function Properties	
Constant Zero or Constant One	% Probability of 1 output
Negated	% Probability of 0 output
Linear	Hamming Weight
Monotonic	Bias
Balanced	Algebraic Degree
Affine	Nonlinearity
Bent	

Cryptographic Boolean Properties


Properties | Correlation

# Variables: 3 # Vectors: 8 # Functions in Family: 256

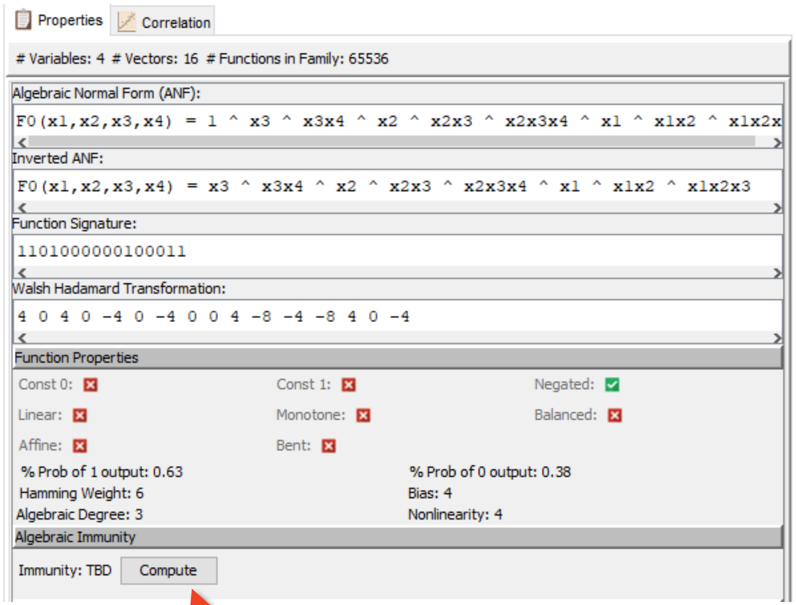
Algebraic Normal Form (ANF):
```
F0(x1,x2,x3) = 1 ^ x2x3 ^ x1 ^ x1x3
```
Inverted ANF:
```
F0(x1,x2,x3) = x2x3 ^ x1 ^ x1x3
```
Function Signature:
```
11100100
```
Walsh Hadamard Transformation:
```
0 0 -4 4 -4 -4 0 0
```
Function Properties

Const 0: ✗ Const 1: ✗ Negated: ✓
Linear: ✗ Monotone: ✗ Balanced: ✓
Affine: ✗ Bent: ✗

% Prob of 1 output: 0.50 % Prob of 0 output: 0.50
Hamming Weight: 4 Bias: 0
Algebraic Degree: 2 Nonlinearity: 2

Algebraic Immunity

Immunity: 3 Compute


Properties | Correlation

# Variables: 4 # Vectors: 16 # Functions in Family: 65536

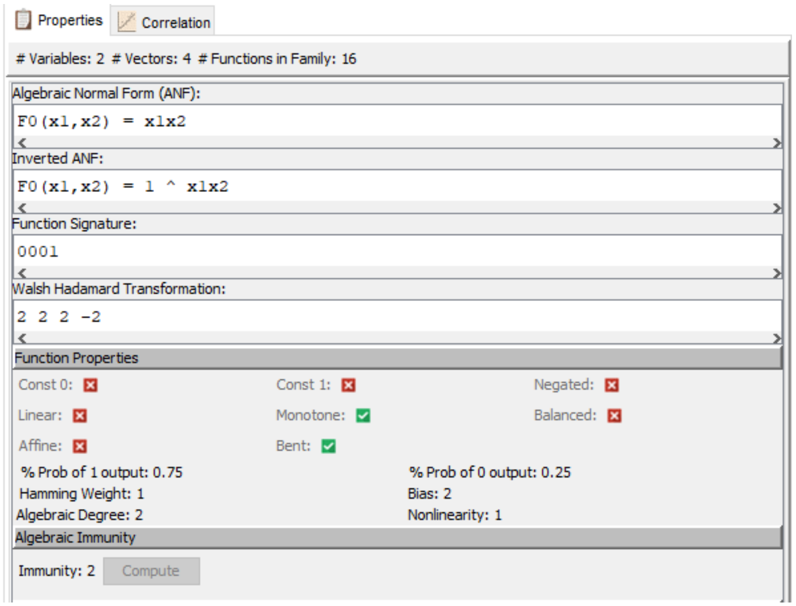
Algebraic Normal Form (ANF):
```
F0(x1,x2,x3,x4) = 1 ^ x3 ^ x3x4 ^ x2 ^ x2x3 ^ x2x3x4 ^ x1 ^ x1x2 ^ x1x2x
```
Inverted ANF:
```
F0(x1,x2,x3,x4) = x3 ^ x3x4 ^ x2 ^ x2x3 ^ x2x3x4 ^ x1 ^ x1x2 ^ x1x2x3
```
Function Signature:
```
1101000000100011
```
Walsh Hadamard Transformation:
```
4 0 4 0 -4 0 -4 0 0 4 -8 -4 -8 4 0 -4
```
Function Properties

Const 0: ✗ Const 1: ✗ Negated: ✓
Linear: ✗ Monotone: ✗ Balanced: ✗
Affine: ✗ Bent: ✗

% Prob of 1 output: 0.63 % Prob of 0 output: 0.38
Hamming Weight: 6 Bias: 4
Algebraic Degree: 3 Nonlinearity: 4

Algebraic Immunity

Immunity: TBD Compute


Properties | Correlation

# Variables: 2 # Vectors: 4 # Functions in Family: 16

Algebraic Normal Form (ANF):
```
F0(x1,x2) = x1x2
```
Inverted ANF:
```
F0(x1,x2) = 1 ^ x1x2
```
Function Signature:
```
0001
```
Walsh Hadamard Transformation:
```
2 2 2 -2
```
Function Properties

Const 0: ✗ Const 1: ✗ Negated: ✗
Linear: ✗ Monotone: ✓ Balanced: ✗
Affine: ✗ Bent: ✓

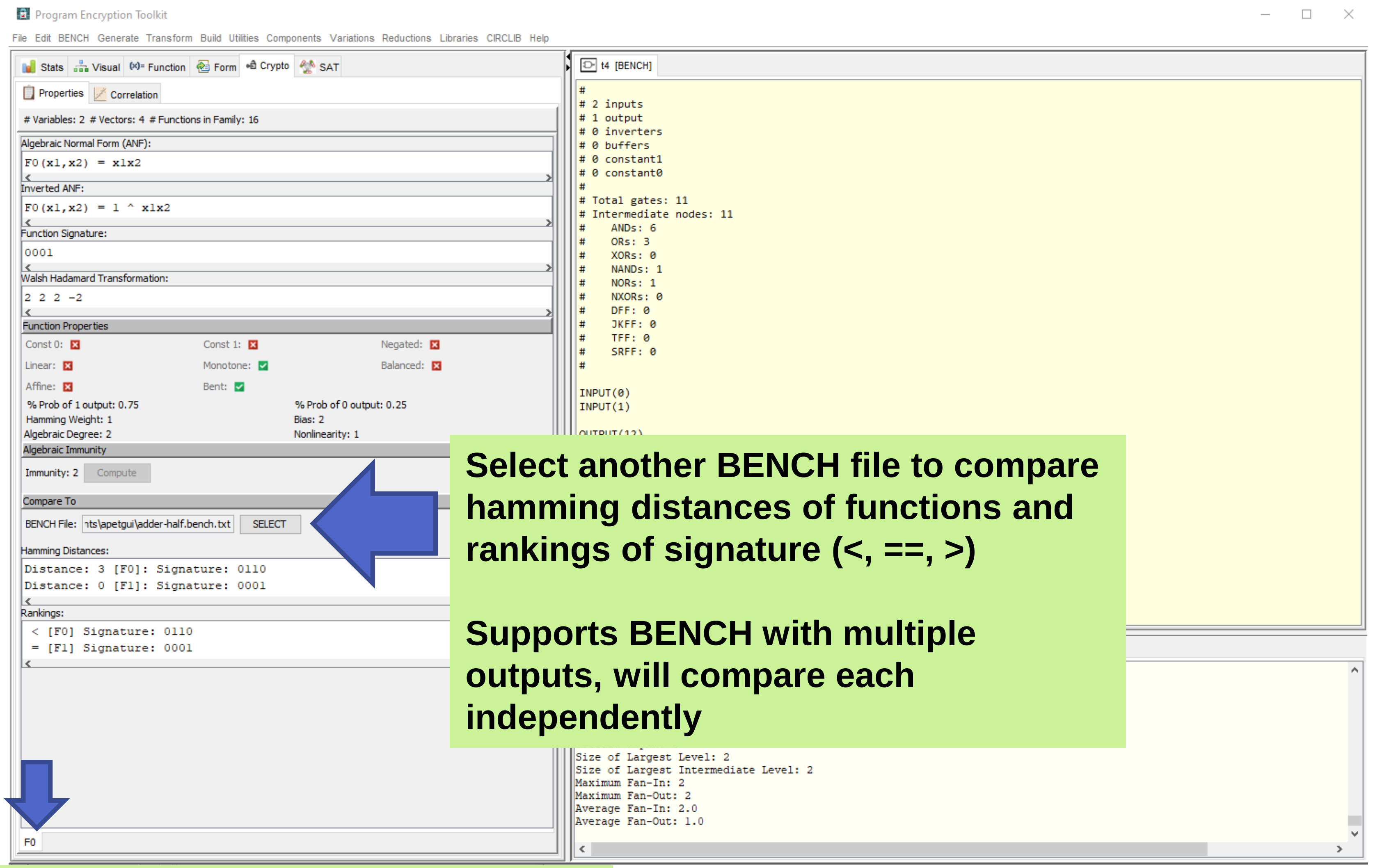
% Prob of 1 output: 0.75 % Prob of 0 output: 0.25
Hamming Weight: 1 Bias: 2
Algebraic Degree: 2 Nonlinearity: 1

Algebraic Immunity

Immunity: 2 Compute

For n=4 inputs, AI has to be explicitly computed

Cryptographic Boolean Properties


Program Encryption Toolkit
File Edit BENCH Generate Transform Build Utilities Components Variations Reductions Libraries CIRCLIB Help
Stats Visual Function Form Crypto SAT
Properties Correlation
t4 [BENCH]
SELECT
Compute
F0

Variables: 2 # Vectors: 4 # Functions in Family: 16

Algebraic Normal Form (ANF):

```
F0(x1,x2) = x1x2
```

Inverted ANF:

```
F0(x1,x2) = 1 ^ x1x2
```

Function Signature:

```
0001
```

Walsh Hadamard Transformation:

```
2 2 2 -2
```

Function Properties

Const 0: ✗	Const 1: ✗	Negated: ✗
Linear: ✗	Monotone: ✓	Balanced: ✗
Affine: ✗	Bent: ✓	

% Prob of 1 output: 0.75 — % Prob of 0 output: 0.25
Hamming Weight: 1 — Bias: 2
Algebraic Degree: 2 — Nonlinearity: 1

Algebraic Immunity

Immunity: 2

Compare To

BENCH File: nts\apetgui\adder-half.bench.txt

Hamming Distances:

```
Distance: 3 [F0]: Signature: 0110
Distance: 0 [F1]: Signature: 0001
```

Rankings:

```
 < [F0] Signature: 0110
 = [F1] Signature: 0001
```

```
#
# 2 inputs
# 1 output
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 11
# Intermediate nodes: 11
#    ANDs: 6
#    ORs: 3
#    XORs: 0
#    NANDs: 1
#    NORs: 1
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)

OUTPUT(12)
```

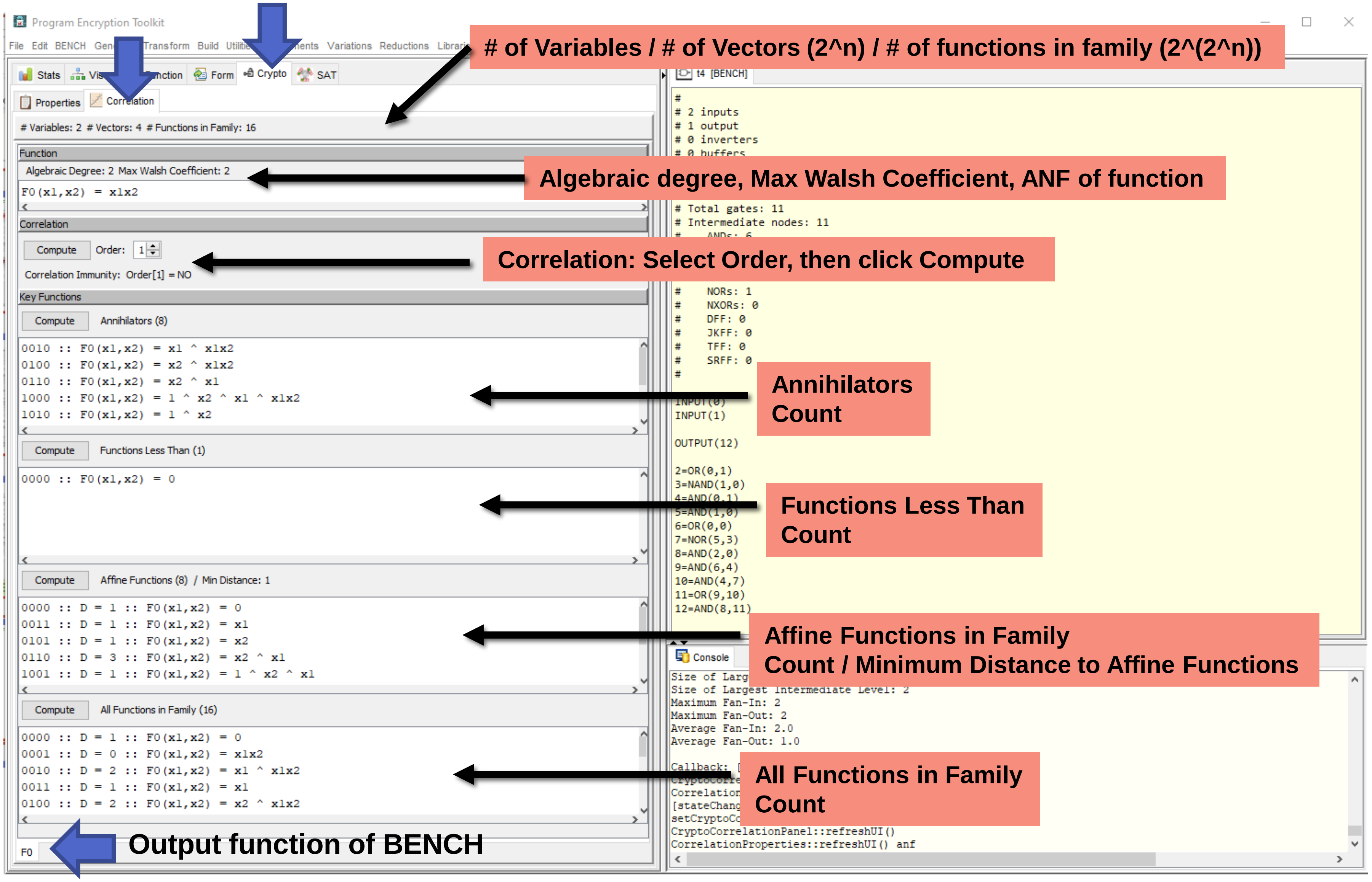
```
Size of Largest Level: 2
Size of Largest Intermediate Level: 2
Maximum Fan-In: 2
Maximum Fan-Out: 2
Average Fan-In: 2.0
Average Fan-Out: 1.0
```

Select another BENCH file to compare hamming distances of functions and rankings of signature (<, ==, >)

Supports BENCH with multiple outputs, will compare each independently

If the BENCH has multiple outputs, each output function gets its own properties and correlation tab

BENCH->Crypto Analysis->Correlation from main menu


Program Encryption Toolkit
File Edit BENCH Gen Transform Build Utiliti ents Variations Reductions Librari
Stats Vis nction Form Crypto SAT
Properties Correlation
t4 [BENCH]
Console
F0

```
# Variables: 2  # Vectors: 4  # Functions in Family: 16

Function
Algebraic Degree: 2  Max Walsh Coefficient: 2
F0(x1,x2) = x1x2

Correlation
Compute   Order: 1
Correlation Immunity: Order[1] = NO

Key Functions
Compute   Annihilators (8)
0010 :: F0(x1,x2) = x1 ^ x1x2
0100 :: F0(x1,x2) = x2 ^ x1x2
0110 :: F0(x1,x2) = x2 ^ x1
1000 :: F0(x1,x2) = 1 ^ x2 ^ x1 ^ x1x2
1010 :: F0(x1,x2) = 1 ^ x2

Compute   Functions Less Than (1)
0000 :: F0(x1,x2) = 0

Compute   Affine Functions (8) / Min Distance: 1
0000 :: D = 1 :: F0(x1,x2) = 0
0011 :: D = 1 :: F0(x1,x2) = x1
0101 :: D = 1 :: F0(x1,x2) = x2
0110 :: D = 3 :: F0(x1,x2) = x2 ^ x1
1001 :: D = 1 :: F0(x1,x2) = 1 ^ x2 ^ x1

Compute   All Functions in Family (16)
0000 :: D = 1 :: F0(x1,x2) = 0
0001 :: D = 0 :: F0(x1,x2) = x1x2
0010 :: D = 2 :: F0(x1,x2) = x1 ^ x1x2
0011 :: D = 1 :: F0(x1,x2) = x1
0100 :: D = 2 :: F0(x1,x2) = x2 ^ x1x2
```

```
#
# 2 inputs
# 1 output
# 0 inverters
# 0 buffers

# Total gates: 11
# Intermediate nodes: 11
#    ANDs: 6

#    NORs: 1
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#
INPUT(0)
INPUT(1)

OUTPUT(12)

2=OR(0,1)
3=NAND(1,0)
4=AND(0,1)
5=AND(1,0)
6=OR(0,0)
7=NOR(5,3)
8=AND(2,0)
9=AND(6,4)
10=AND(4,7)
11=OR(9,10)
12=AND(8,11)
```

```
Size of Larg
Size of Largest Intermediate Level: 2
Maximum Fan-In: 2
Maximum Fan-Out: 2
Average Fan-In: 2.0
Average Fan-Out: 1.0

Callback: [
CryptoCorre
Correlation
[stateChang
setCryptoCo
CryptoCorrelationPanel::refreshUI()
CorrelationProperties::refreshUI() anf
```

# of Variables / # of Vectors (2^n) / # of functions in family (2^(2^n))

Algebraic degree, Max Walsh Coefficient, ANF of function

Correlation: Select Order, then click Compute

Annihilators Count

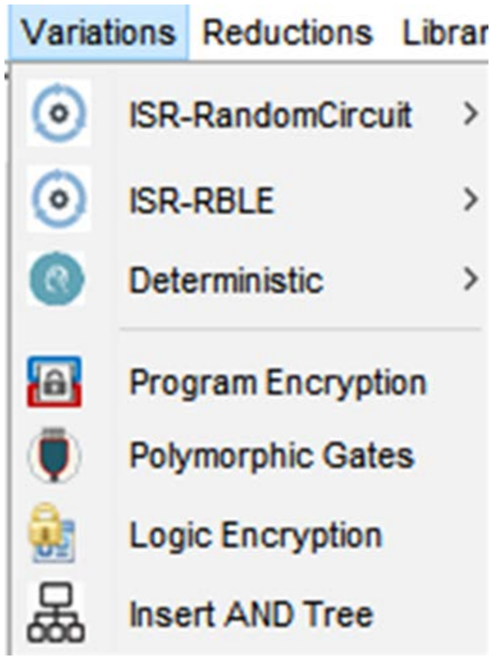
Functions Less Than Count

Affine Functions in Family Count / Minimum Distance to Affine Functions

All Functions in Family Count

Output function of BENCH

Circuit Variations:



- Iterative Selection/Replacement
  - Random Circuit
  - Random Boolean Logic Expansion
- Deterministic
  - Boundary Blur
  - Component Fusion
  - Component Encryption
- Program Encryption
- Polymorphic Gates
- Logic Encryption
- Insert AND Tree

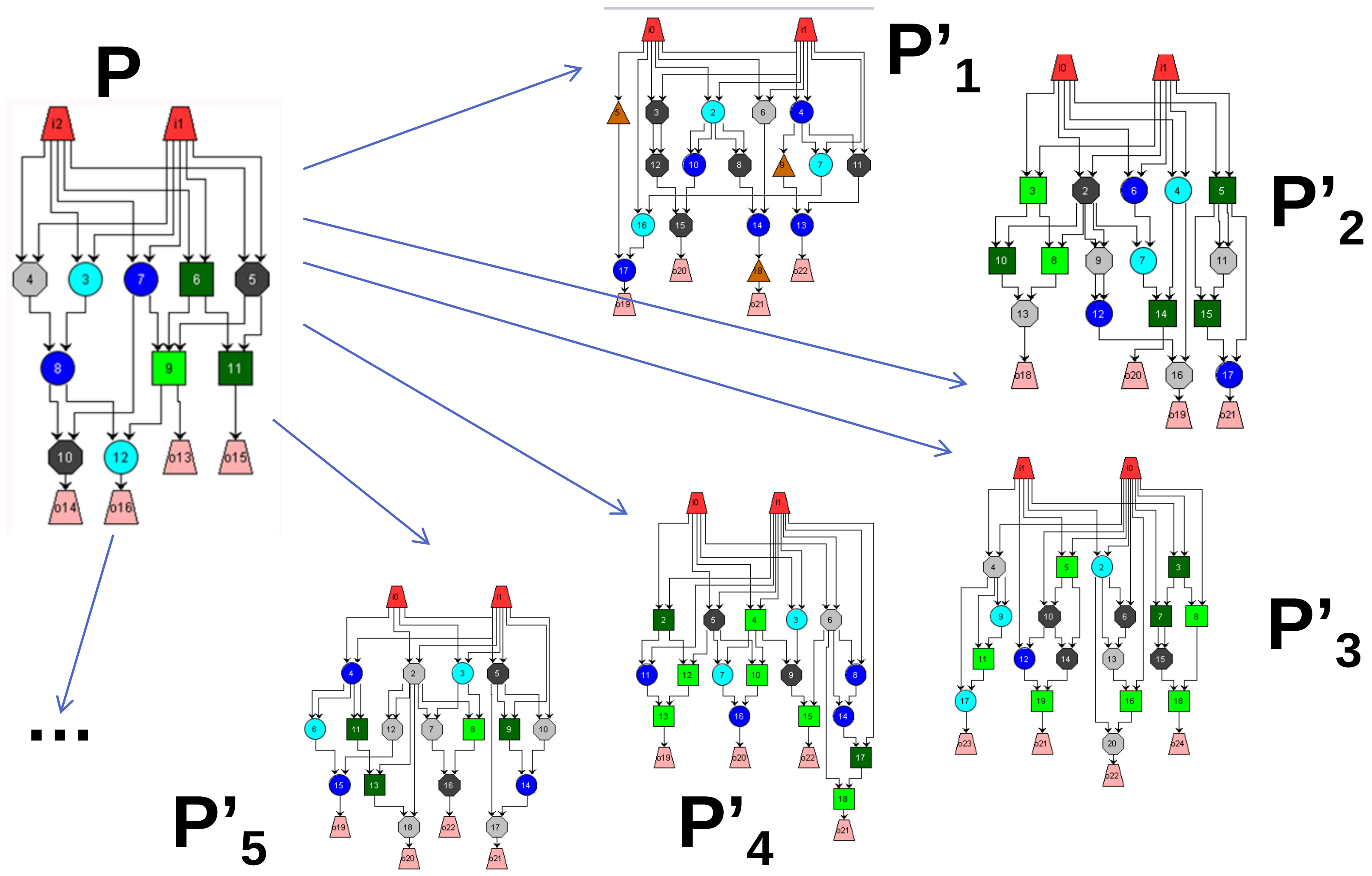
- Utilities->Permutation Circuits

Variation Techniques

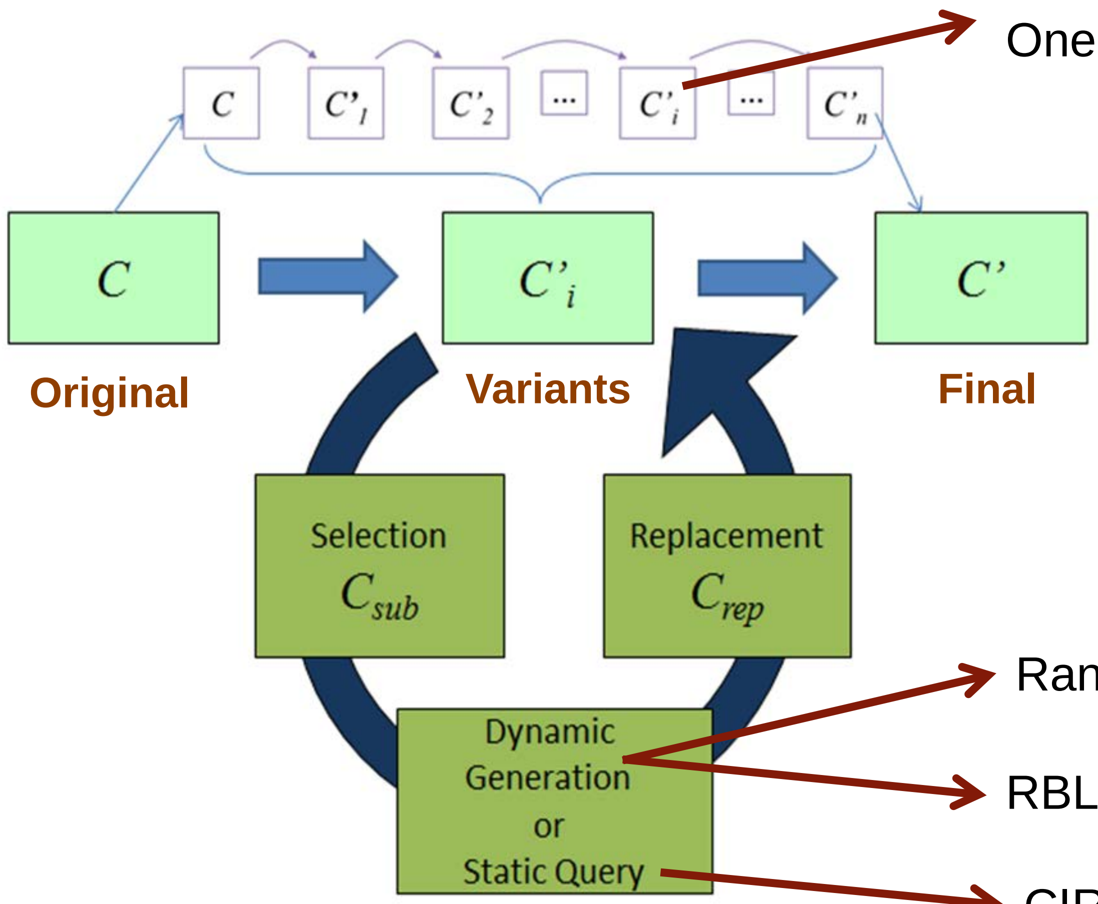
- **Random** variation techniques may (or may not) hide certain design information
- **Deterministic** variation techniques are geared at **hiding components**
  - Boundary Blurring
  - Component Fusion
  - Component Encryption
- Some variation techniques can hide the complete design elements of a circuit
  - Virtual Black Box (Synthesis)
  - Polymorphic Gates / Functional Polymorphism
- Some variation techniques can hide the **full function** of a circuit, if the circuit input size is small enough
  - Program Encryption

Structural Polymorphism

- **Structural Polymorphic** generation is easy…
  - **ONE FUNCTION, MANY FORMS…**
  - Combinational logic = straight-line program code / basic blocks



Iterative Selection/Replacement



One "**Iteration**"

Round: when all gates of a circuit have been replaced through selection/replacement operations

Random Circuit Generation

RBLE Generation

CIRCLIB Library Selection

- Three primary options:
  - Iteration based
  - Round based
  - Size based

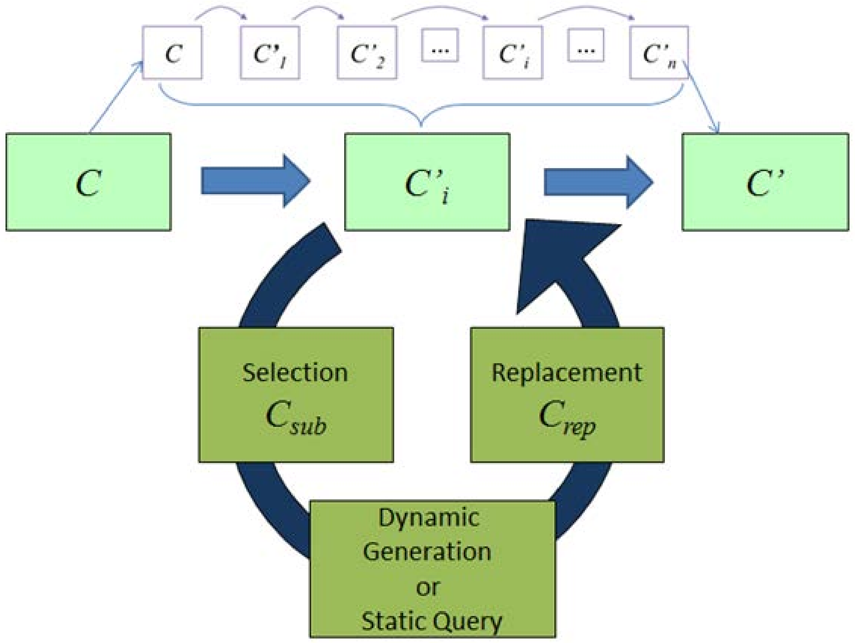
These options currently ALL use random circuit generation for the replacement step

Picking a Random Circuit with Function *f()*

- **The problem:** Given a circuit C_{sub}, **pick** a suitable replacement C_{rep} for it from the same family (same input/output size), but with larger (or smaller) gate size, and that does the same function as C_{sub} (semantic equivalence)

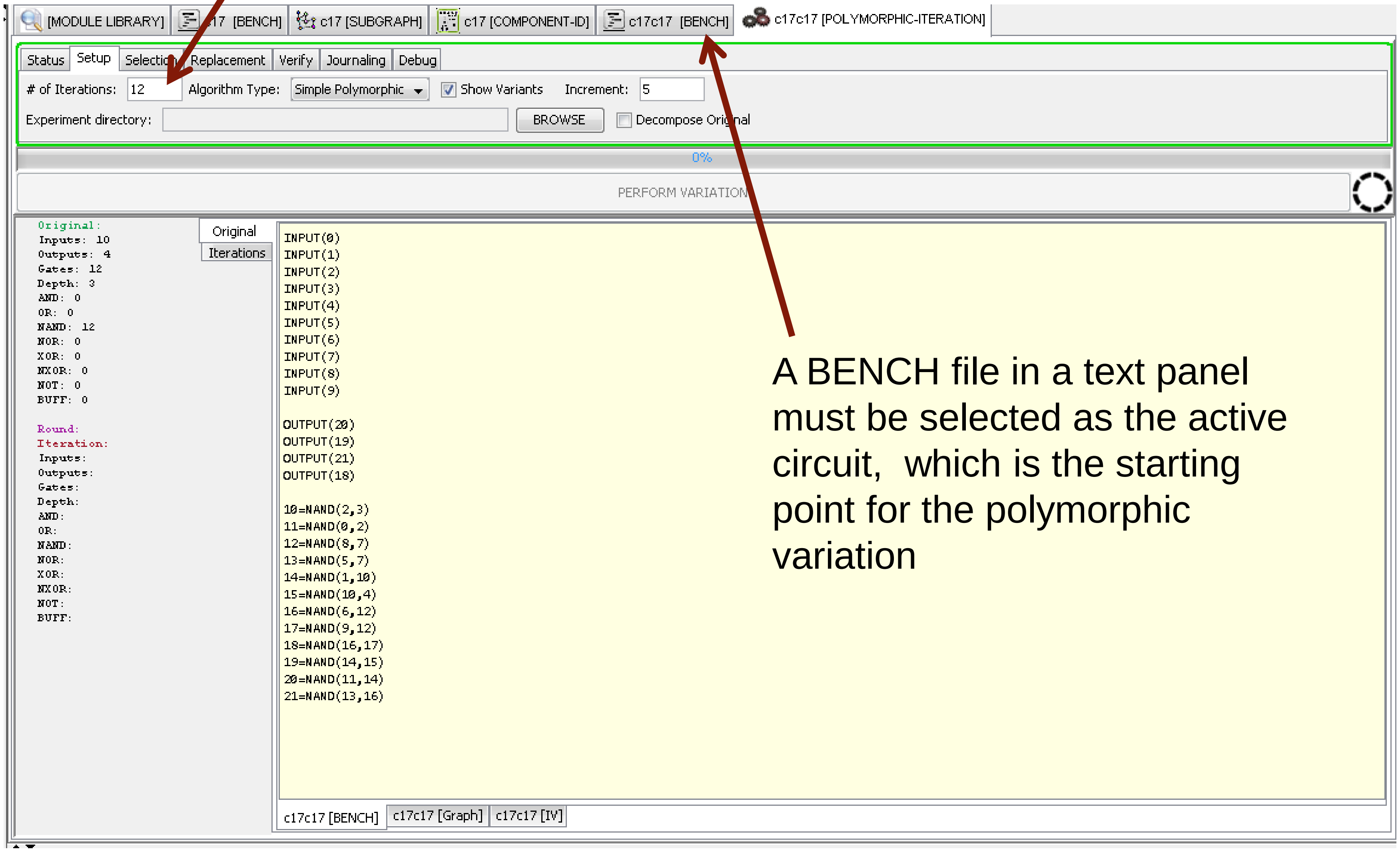
- *If we limit C_{sub} to be small (in gate size), we could iteratively repeat this process of selecting and replacing subcircuits in a larger circuit C*

Iterative Sub-Circuit Selection and Replacement (ISR)



Iteration Based

- Create a variant based on a fixed # of iterations (selection/replacement increments)



# of Iterations: 12 Algorithm Type: Simple Polymorphic ☑ Show Variants Increment: 5

Experiment directory: ☐ Decompose Original

```
Original:
Inputs: 10
Outputs: 4
Gates: 12
Depth: 3
AND: 0
OR: 0
NAND: 12
NOR: 0
XOR: 0
NXOR: 0
NOT: 0
BUFF: 0

Round:
Iteration:
Inputs:
Outputs:
Gates:
Depth:
AND:
OR:
NAND:
NOR:
XOR:
NXOR:
NOT:
BUFF:
```

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(20)
OUTPUT(19)
OUTPUT(21)
OUTPUT(18)

10=NAND(2,3)
11=NAND(0,2)
12=NAND(8,7)
13=NAND(5,7)
14=NAND(1,10)
15=NAND(10,4)
16=NAND(6,12)
17=NAND(9,12)
18=NAND(16,17)
19=NAND(14,15)
20=NAND(11,14)
21=NAND(13,16)
```

A BENCH file in a text panel must be selected as the active circuit, which is the starting point for the polymorphic variation

Iteration Based

Basic walk-through: after picking # of iterations

1. Pick a general selection algorithm type: where or how gates are selected within the circuit

    Simple Polymorphic (uses random gate)

    Random Level

    Output Level

    Largest Level

    Smallest Level

    Fixed Level

2. Show variants and increment #: Create a panel that shows the BENCH and graph of the variant within the GUI.

Displays variants based on increment # (i.e., every 1, every 2, every 5, every 10, etc.)

Iteration Based

Basic walk-through:

3. Choose an experiment directory (BROWSE)

Original, final, and incremental files will be placed in here

4. Choose whether the original circuit should have decomposed fan-in gates (fan-in will be 2 for all gates if this option is selected)

5. Selection Tab: choose a minimum and maximum selection size (if equal, then only selections of that size are considered)

6. Selection | Smart Strategy: Smart selection will keep track of original gates in the circuit and make future selections from gates that have not been replaced yet

Iteration Based

Basic walk-through:

7. Selection | Use random selection algorithm: can override the general selection algorithm chosen to pick a random method each iteration

8. Selection | Maximum selection attempts: based on available gates for selection and the algorithm chosen, it may not be possible to select a subcircuit of a given size.
    - This is because some selections, when a replacement circuit is inserted, may induce a cycle in the circuit
    - After the max selection attempts are reached, a new selection strategy is chosen (typically, pick 1 or 2 random gates)

9. Selection | Maximum selection input size: selection size is normally based on # of gates, but you can restrict how many inputs a resulting subcircuit can have with this option

Iteration Based

Basic walk-through:

10. Selection | Target Level: only enabled for Fixed Level selection algorithm

11. Replacement Tab: Replacement size is the primary driver. For a given selected subcircuit, sets the target size of the replacement circuit. Based on the random generator, the size winds up being multiplied per output of the selected subcircuit. Depending on how many common inputs the circuit size has, it could be less.

12. Replacement | Basis Set: For replacement circuits, sets the kind of gates that are allowed in the circuit.

Iteration Based

Basic walk-through:

13. Replacement | Use Smart Random: tells the circuit generator to weed out circuits with redundant logic (dual fan-in gates, repeated gates, etc)

14. Replacement | Max Fan-In: tells the circuit generator how many fan-ins a gate might be allowed

15. Replacement | Max Generation Attempts: if the circuit generator cannot find a replacement within a certain # of generation attempts, the algorithm will abandon the selection and pick a new selected subcircuit (typically happens with larger input/output size subcircuits)

Iteration Based

Basic walk-through:

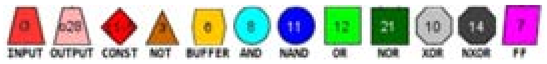
16. Verify Tab: has options for verifying variants. For larger circuits, and Input Vector can be used.

17. Journaling Tab: Saves original, final, and/or intermediate variant files in the experiment directory

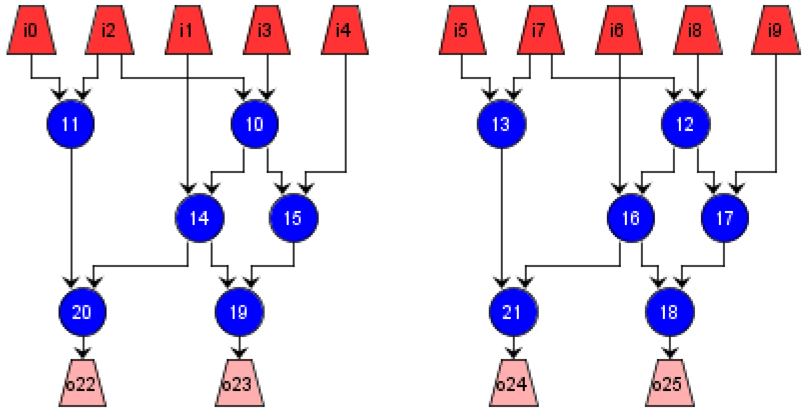
- For variants, increment value specifies how often (how many iterations) to go before saving files (every 1, every 2, etc).
- Journal options: each time a save is done, which circuit formats should be saved => BENCH, GraphML, Hierarchical Image, Organic Image, VHDL, UW format
- File naming is handled automatically

17. Debug Tab: Various options for verbose output to the console as the generation process is executing

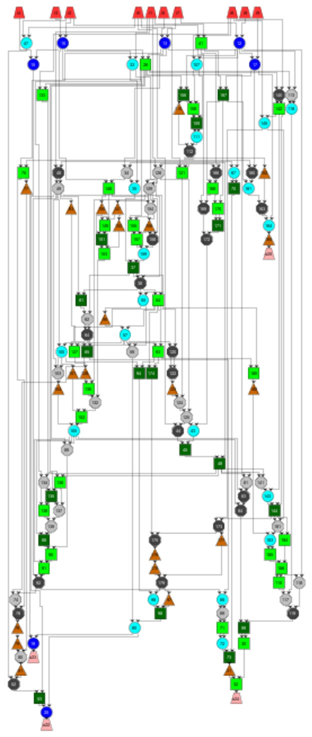
Iteration Based

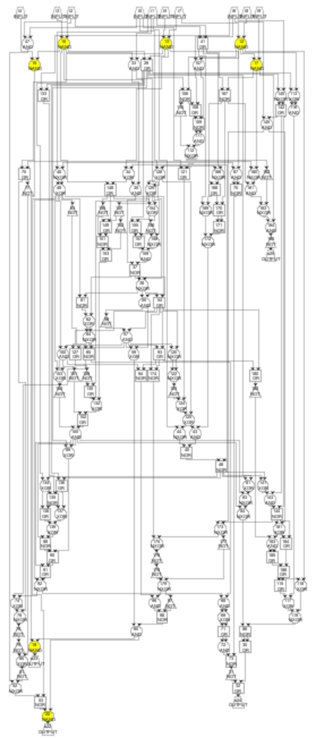


Example: Original Circuit



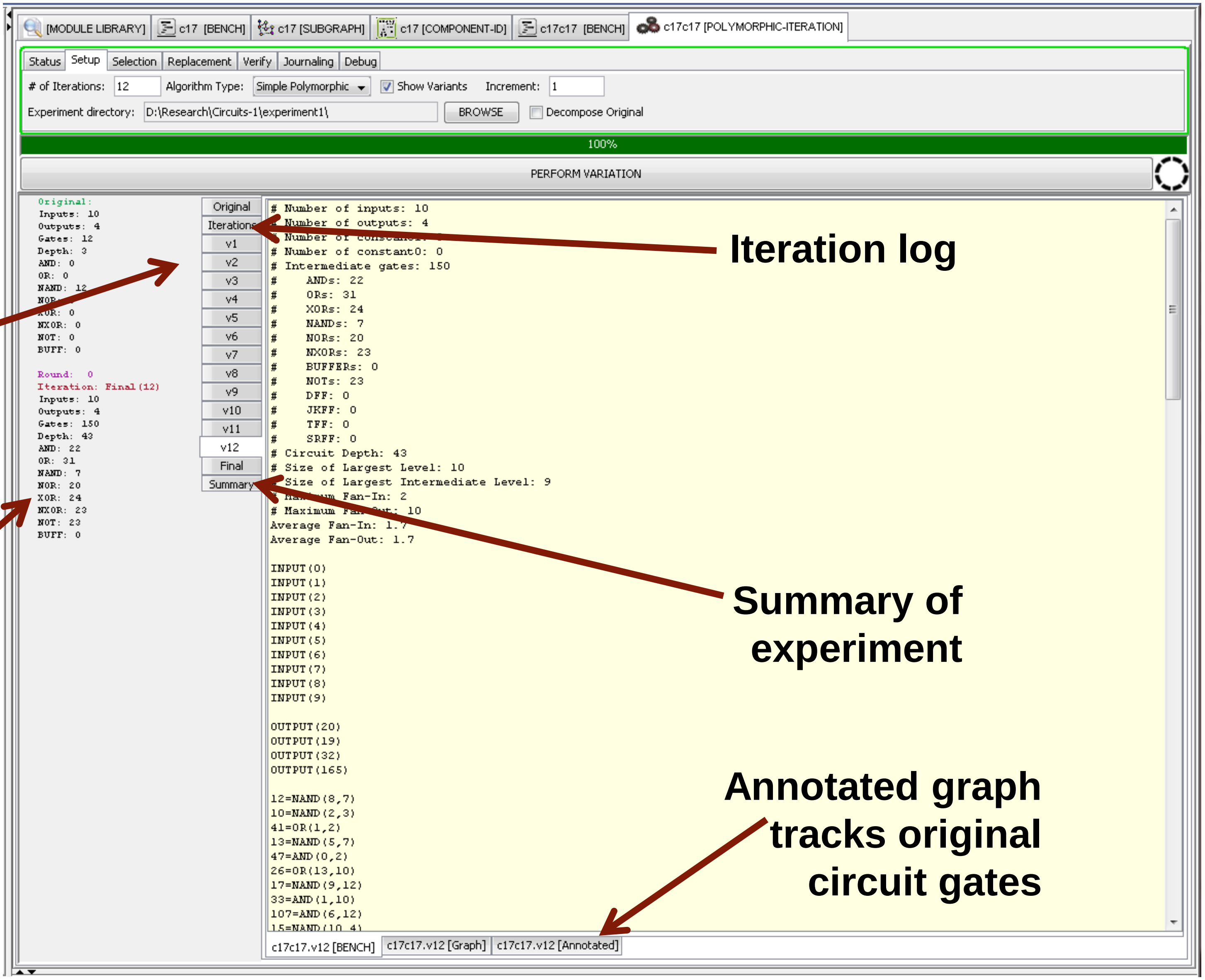
Options:
12 Iterations
Algorithm: Simple polymorphic
Show Variants: yes, increment = 1
Selection size min/max = 2
No smart selection
Max selection input size = 4
Replacement Size = 7
No Smart Random
Max Fan-in = 2
Basis = NOR, AND, OR, XOR, NXOR, NOT
Journal: Final, Original,
Variant w/ Increment = 1


Graph of circuit variant


Annotated graph tracks original circuit gates

Iteration Based

Options:
12 Iterations
Algorithm: Simple polymorphic
Show Variants: yes, increment = 1
Selection size min/max = 2
No smart selection
Max selection input size = 4
Replacement Size = 7
No Smart Random
Max Fan-in = 2
Basis = NOR, AND, OR, XOR, NXOR, NOT
Journal: Final, Original,
Variant w/ Increment = 1



# of Iterations: 12 Algorithm Type: Simple Polymorphic [✓] Show Variants Increment: 1
Experiment directory: D:\Research\Circuits-1\experiment1\ [ ] Decompose Original

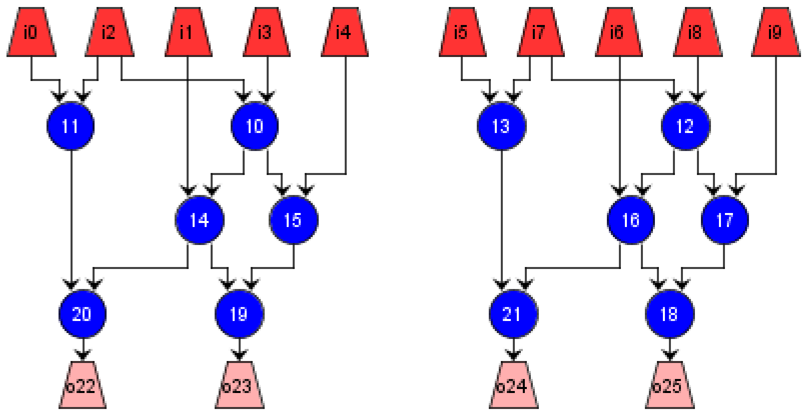
```
Original:
Inputs: 10
Outputs: 4
Gates: 12
Depth: 3
AND: 0
OR: 0
NAND: 12
NOR: [illegible]
XOR: 0
NXOR: 0
NOT: 0
BUFF: 0

Round:  0
Iteration: Final(12)
Inputs: 10
Outputs: 4
Gates: 150
Depth: 43
AND: 22
OR: 31
NAND: 7
NOR: 20
XOR: 24
NXOR: 23
NOT: 23
BUFF: 0
```

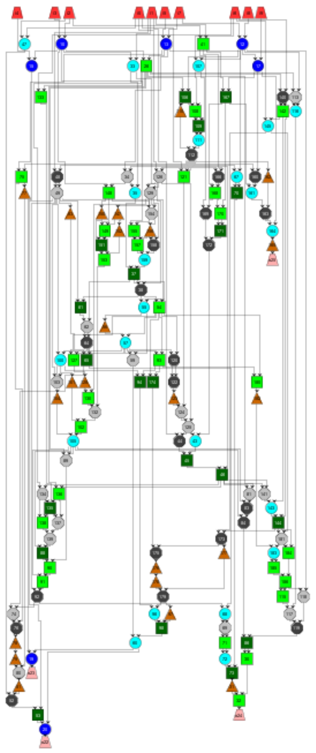
```
# Number of inputs: 10
# Number of outputs: 4
# Number of constant1: 0
# Number of constant0: 0
# Intermediate gates: 150
#    ANDs: 22
#    ORs: 31
#    XORs: 24
#    NANDs: 7
#    NORs: 20
#    NXORs: 23
#    BUFFERs: 0
#    NOTs: 23
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
# Circuit Depth: 43
# Size of Largest Level: 10
# Size of Largest Intermediate Level: 9
# Maximum Fan-In: 2
# Maximum Fan-Out: 10
Average Fan-In: 1.7
Average Fan-Out: 1.7

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)

OUTPUT(20)
OUTPUT(19)
OUTPUT(32)
OUTPUT(165)

12=NAND(8,7)
10=NAND(2,3)
41=OR(1,2)
13=NAND(5,7)
47=AND(0,2)
26=OR(13,10)
17=NAND(9,12)
33=AND(1,10)
107=AND(6,12)
15=NAND(10,4)
```

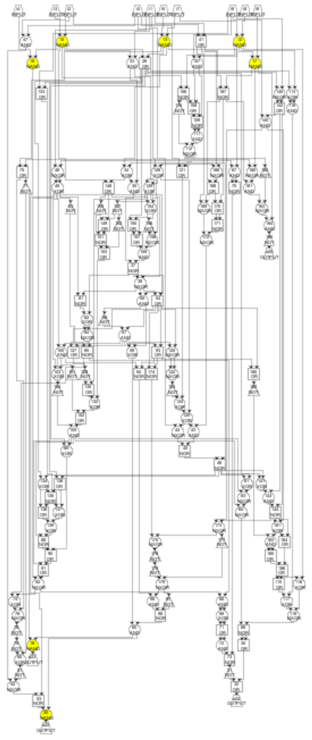
Iteration Based

INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Example: Original Circuit







Options:

12 Iterations

Algorithm: Simple polymorphic

Show Variants: yes, increment = 1

Selection size min/max = 2

No smart selection

Max selection input size = 4

Replacement Size = 7

No Smart Random

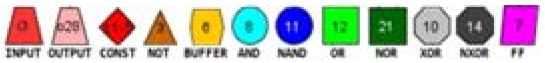
Max Fan-in = 2

Basis = NOR, AND, OR, XOR, NXOR, NOT

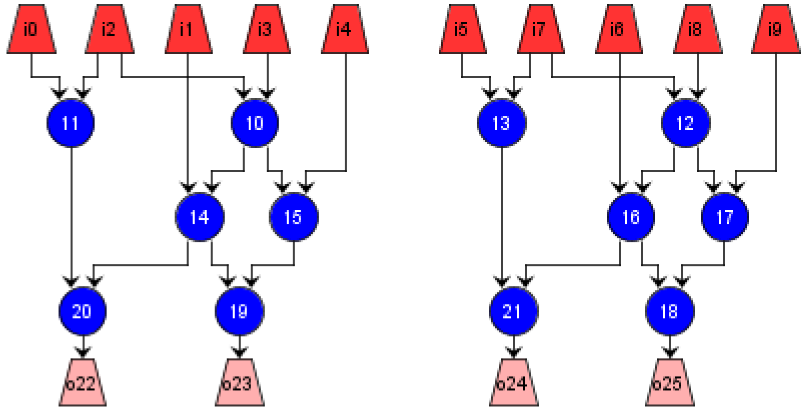
Journal: Final, Original,

Variant w/ Increment = 1

Size Based

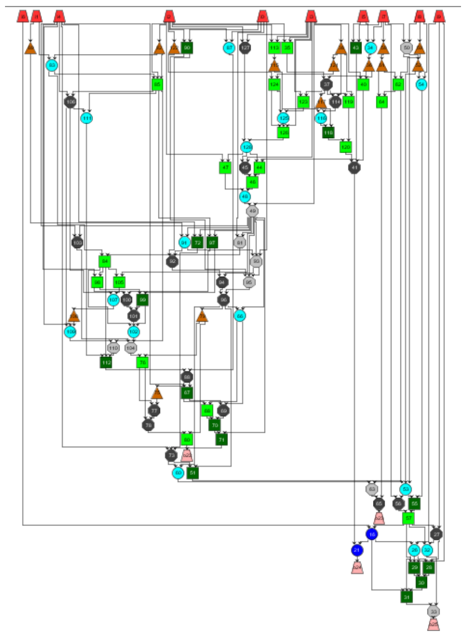


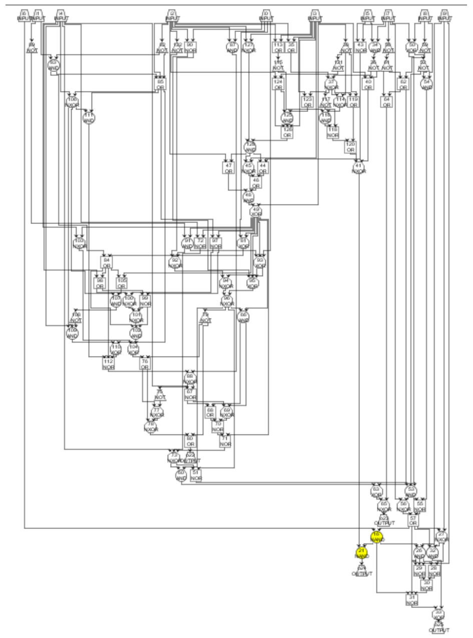
Example: Original Circuit



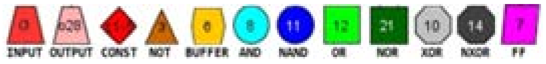
Options:

Size = 100
Algorithm: Simple polymorphic
Show Variants: yes, increment = 5
Selection size min/max = 2
Use smart selection
Max selection input size = 6
Replacement Size = 8
No Smart Random
Max Fan-in = 3
Basis = NOR, AND, OR, XOR, NXOR, NOT
Journal: Final, Original,
Variant w/ Increment = 1

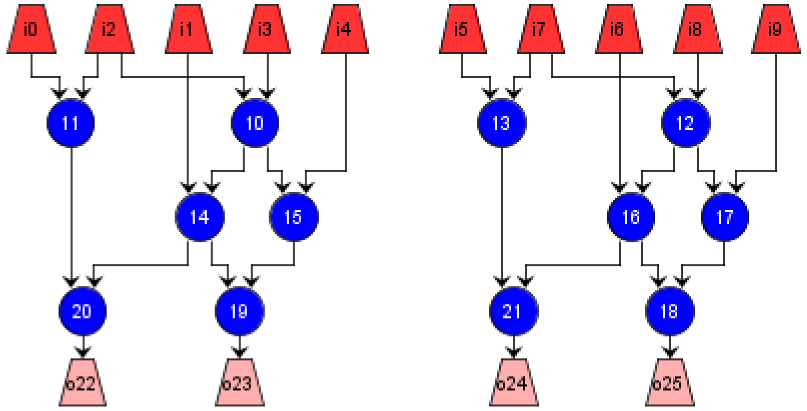




Round Based

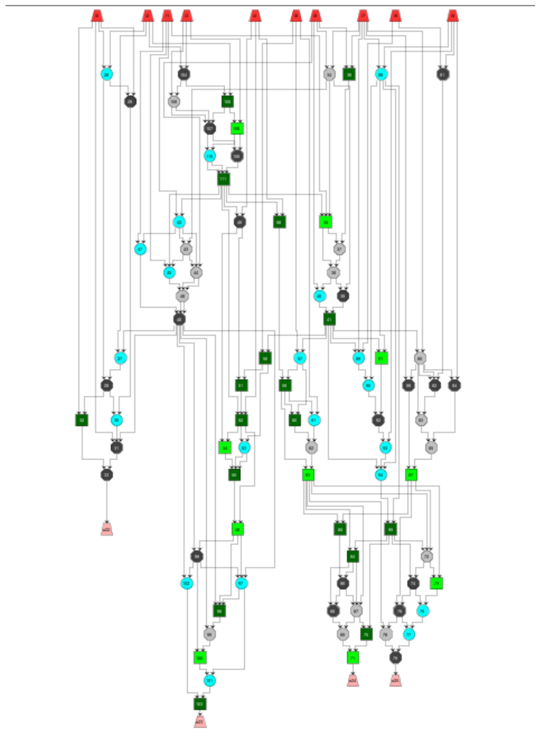


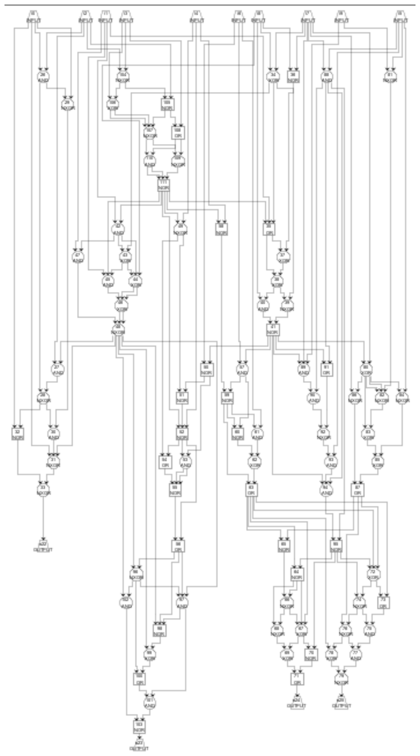
Example: Original Circuit



Options:

Round = 1
Algorithm: Simple polymorphic
Show Variants: yes, increment = 1
Selection size min/max = 2
Use smart selection
Max selection input size = 6
Replacement Size = 8
No Smart Random
Max Fan-in = 3
Basis = NOR, AND, OR, XOR, NXOR, NOT
Journal: Final, Original,
Variant w/ Increment = 1

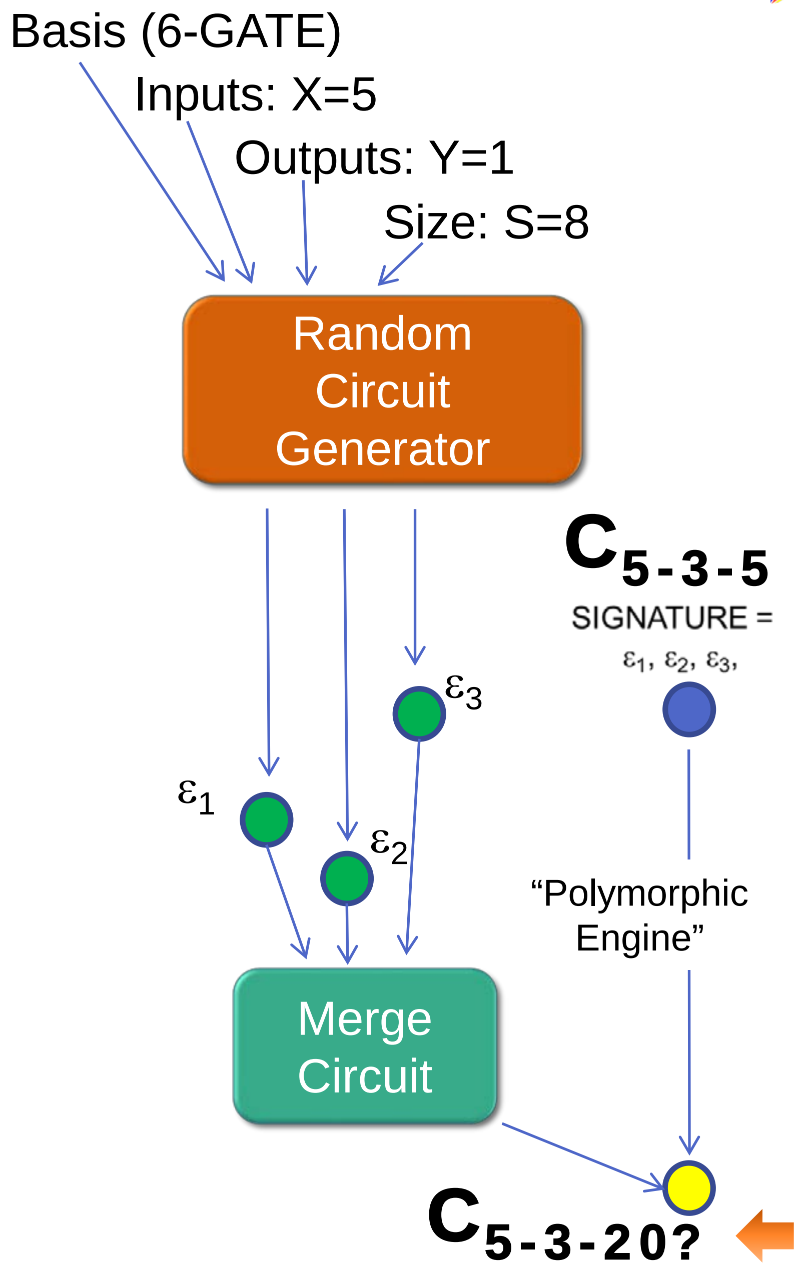




Replacement with Random vs RBLE Generation

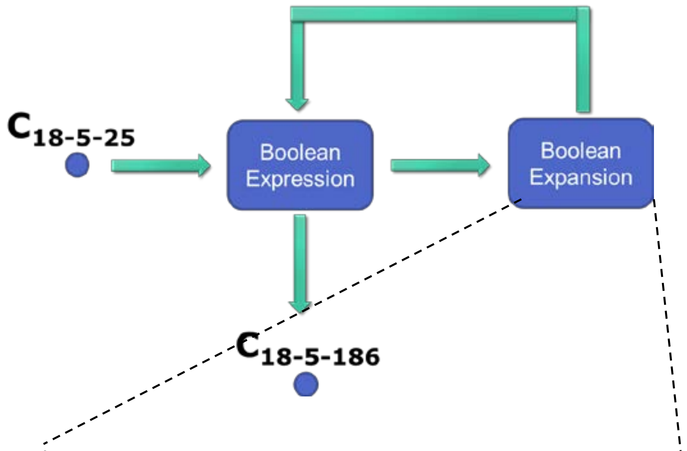
- **Random Circuit Generation:**
  - Given a subcircuit C, random generation will take the input/output size of the subcircuit and generate a random circuit with that IO size and some gate size
  - Randomly generated circuits are compared against the truth table (signature) of the input circuit C until match is found
  - Current engine decomposes multi-output functions and generates a random circuit for each function
  - Single function circuits are merged backed together to produce the final replacement circuit
  - Random circuit generation is non-deterministic in terms of generating a semantically equivalent circuit in a tractable time limit
  - *prob(sig(C) == sig(R_x))* related to statistical distribution of circuits in a family with a given signature
  - Probability of random circuit with matching signature is of the order, where *n* is the number of inputs and *m* is the number of outputs:

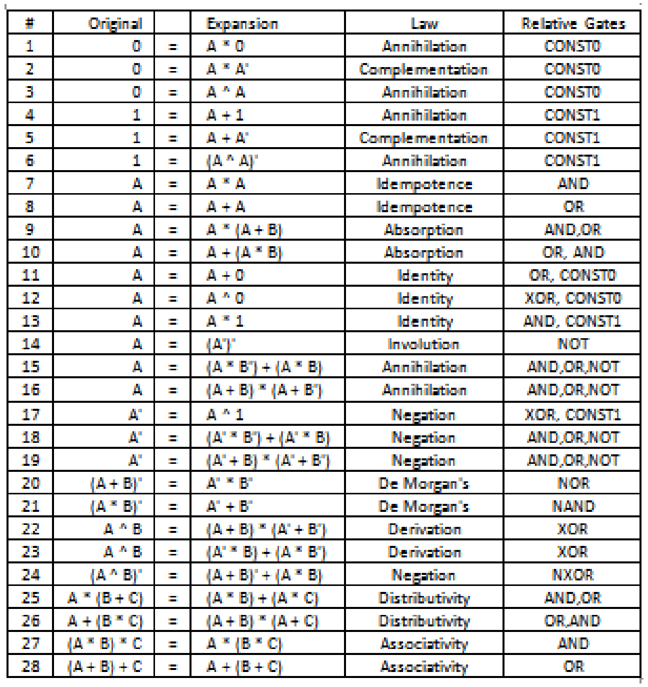
    $$\mathbf{m(1 / 2^{2^{\wedge}n})}$$



Replacement with Random vs RBLE Generation

- **Random Boolean Logic Expansion (RBLE):**
  - Given a subcircuit C, RBLE will take the existing circuit structure and express it as a Boolean logic expression
  - Inverse of Boolean logic laws are applied in some random fashion until given constraints of the replacement circuit are met
  - Applying logic laws inversely produces “expansion” vs. “reduction” of the logic expression
  - Expansions are applied repeatedly on the Boolean logic expression
  - Three possible generation policies are defined
    - Fixed: deterministic/most efficient runtime
    - Strict Size: nondeterministic/most precise replacement circuit size
    - Target Size: nondeterministic/





#	Original		Expansion	Law	Relative Gates
1	0	=	A * 0	Annihilation	CONST0
2	0	=	A * A'	Complementation	CONST0
3	0	=	A ^ A	Annihilation	CONST0
4	1	=	A + 1	Annihilation	CONST1
5	1	=	A + A'	Complementation	CONST1
6	1	=	(A ^ A)'	Annihilation	CONST1
7	A	=	A * A	Idempotence	AND
8	A	=	A + A	Idempotence	OR
9	A	=	A * (A + B)	Absorption	AND,OR
10	A	=	A + (A * B)	Absorption	OR, AND
11	A	=	A + 0	Identity	OR, CONST0
12	A	=	A ^ 0	Identity	XOR, CONST0
13	A	=	A * 1	Identity	AND, CONST1
14	A	=	(A')'	Involution	NOT
15	A	=	(A * B') + (A * B)	Annihilation	AND,OR,NOT
16	A	=	(A + B) * (A + B')	Annihilation	AND,OR,NOT
17	A'	=	A ^ 1	Negation	XOR, CONST1
18	A'	=	(A' * B') + (A' * B)	Negation	AND,OR,NOT
19	A'	=	(A' + B) * (A' + B')	Negation	AND,OR,NOT
20	(A + B)'	=	A' * B'	De Morgan's	NOR
21	(A * B)'	=	A' + B'	De Morgan's	NAND
22	A ^ B	=	(A + B) * (A' + B')	Derivation	XOR
23	A ^ B	=	(A' * B) + (A * B')	Derivation	XOR
24	(A ^ B)'	=	(A + B)' + (A * B)	Negation	NXOR
25	A * (B + C)	=	(A * B) + (A * C)	Distributivity	AND,OR
26	A + (B * C)	=	(A + B) * (A + C)	Distributivity	OR,AND
27	(A * B) * C	=	A * (B * C)	Associativity	AND
28	(A + B) + C	=	A + (B + C)	Associativity	OR

RBLE Example

g1 = (i0 * i1)'

1: (0 + (i0 * i1))’

2: ((i1' * 0) + (i1 * i0))’

3: ((i1' * (i0 * i0')) + (i1 * i0))'

4: ((i1' * (i0 * i0')) + (i1 * (i0 * i0)))’

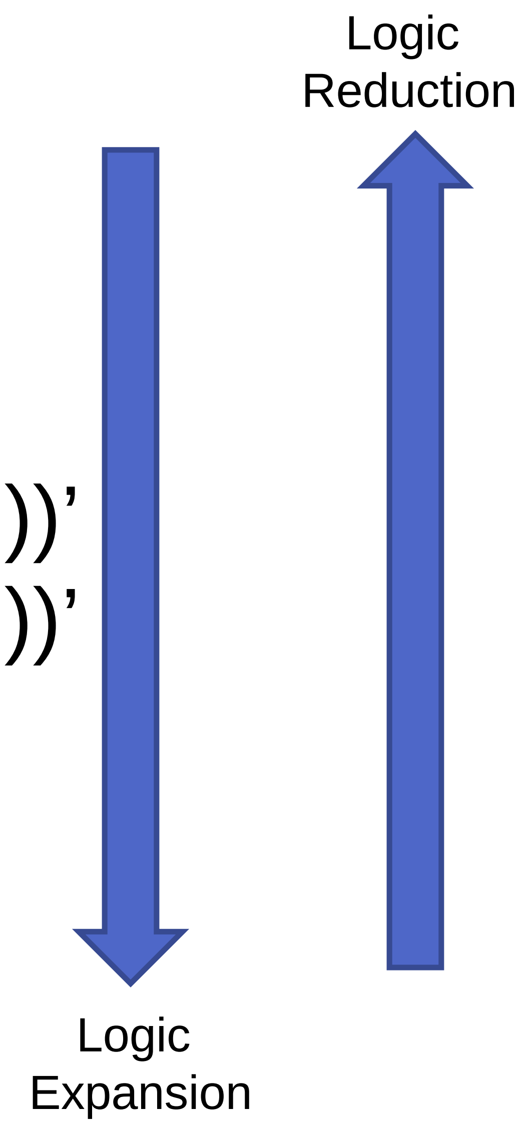
5: ((i1' * (i0 * i0')) + (i0 * (i1 * i0)))’

6: ((i0 * (i1'*i0')) + (i0 * (i1*i0)))’

7: (((i1' * i0') + (i1 * i0)) * i0)’

8: ((i1 ^ i0)' * i0)’

g1 = ((i1 ^ i0)' * i0)‘

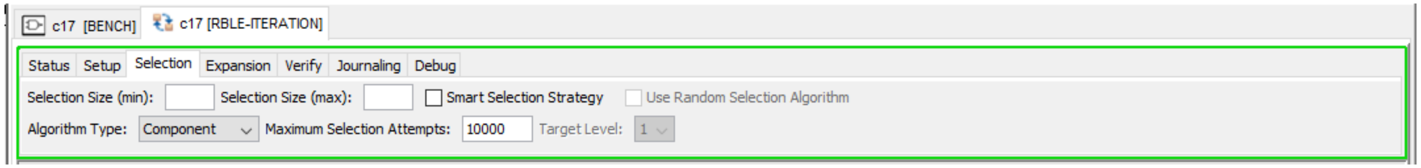


RBLE Generation Policies

- **Fixed:** Apply a “fixed” number of expansions
  - Deterministic/always returns a variant
  - Most efficient/linear runtime
  - Size of ISR variant unpredictable
- **Strict Size:** Apply expansions until a gate size *n* is reached
  - Non-deterministic/may fail to return a variant (max attempts)
  - Requires trials, which increase run-time (max expansions)
  - Allows precise ISR size estimation
- **Target Size:** Apply expansions until a target get size *n* is reached or exceeded
  - Non-deterministic/may fail to return a variant (max attempts),
  - Requires trials, which increase run-time (max expansions)
  - Allows more accurate ISR size estimation

RBLE Options

Selection options are the same as for random circuit generation:



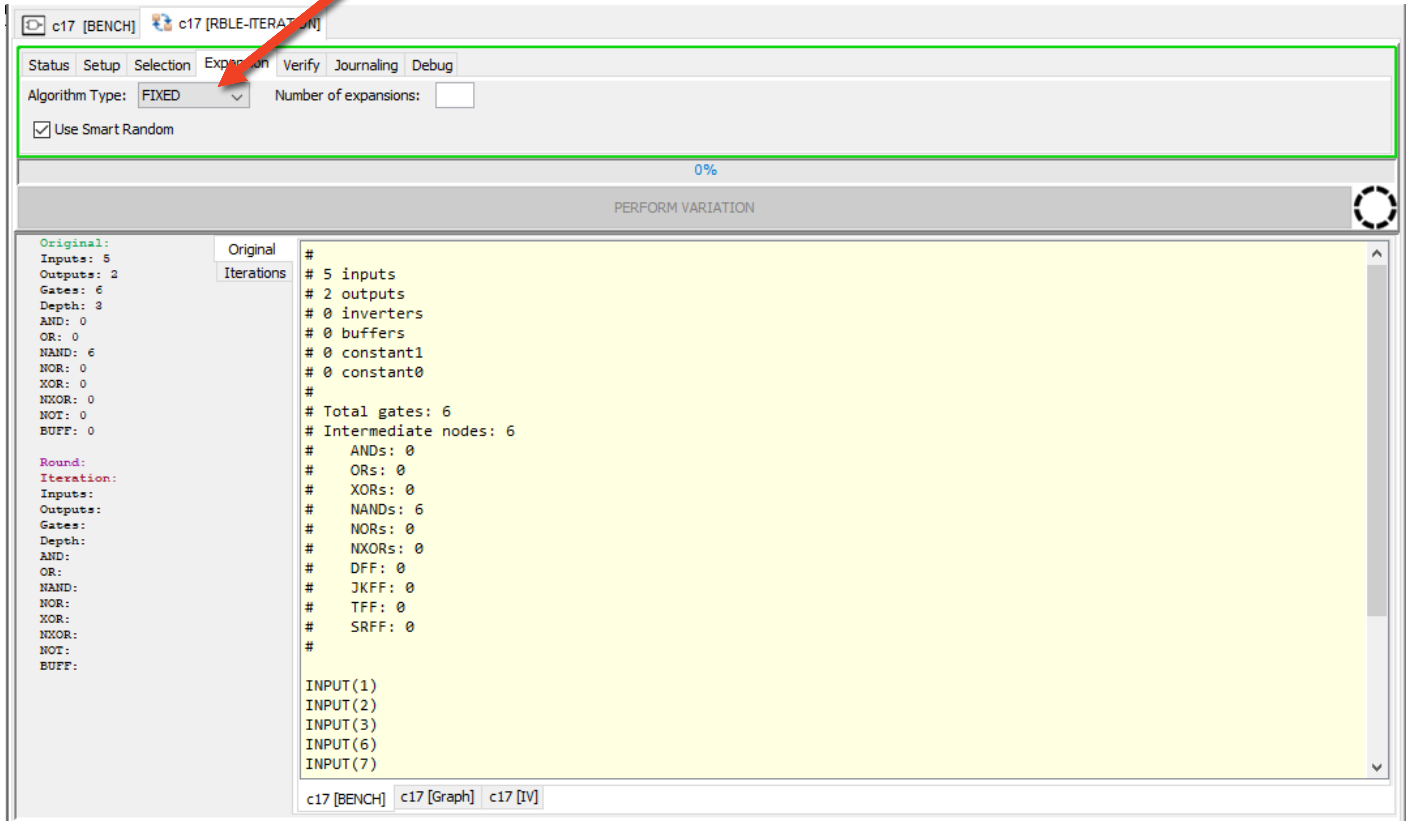
c17 [BENCH] | c17 [RBLE-ITERATION]

Status | Setup | Selection | Expansion | Verify | Journaling | Debug

Selection Size (min): Selection Size (max): ☐ Smart Selection Strategy ☐ Use Random Selection Algorithm

Algorithm Type: Component Maximum Selection Attempts: 10000 Target Level: 1

Expansion options select FIXED, STRICT, or TARGET



c17 [BENCH] | c17 [RBLE-ITERATION]

Status | Setup | Selection | Expansion | Verify | Journaling | Debug

Algorithm Type: FIXED Number of expansions:

☑ Use Smart Random

0%

PERFORM VARIATION

```
Original:
Inputs: 5
Outputs: 2
Gates: 6
Depth: 3
AND: 0
OR: 0
NAND: 6
NOR: 0
XOR: 0
NXOR: 0
NOT: 0
BUFF: 0

Round:
Iteration:
Inputs:
Outputs:
Gates:
Depth:
AND:
OR:
NAND:
NOR:
XOR:
NXOR:
NOT:
BUFF:
```

Original | Iterations

```
#
# 5 inputs
# 2 outputs
# 0 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 6
# Intermediate nodes: 6
#    ANDs: 0
#    ORs: 0
#    XORs: 0
#    NANDs: 6
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)
```

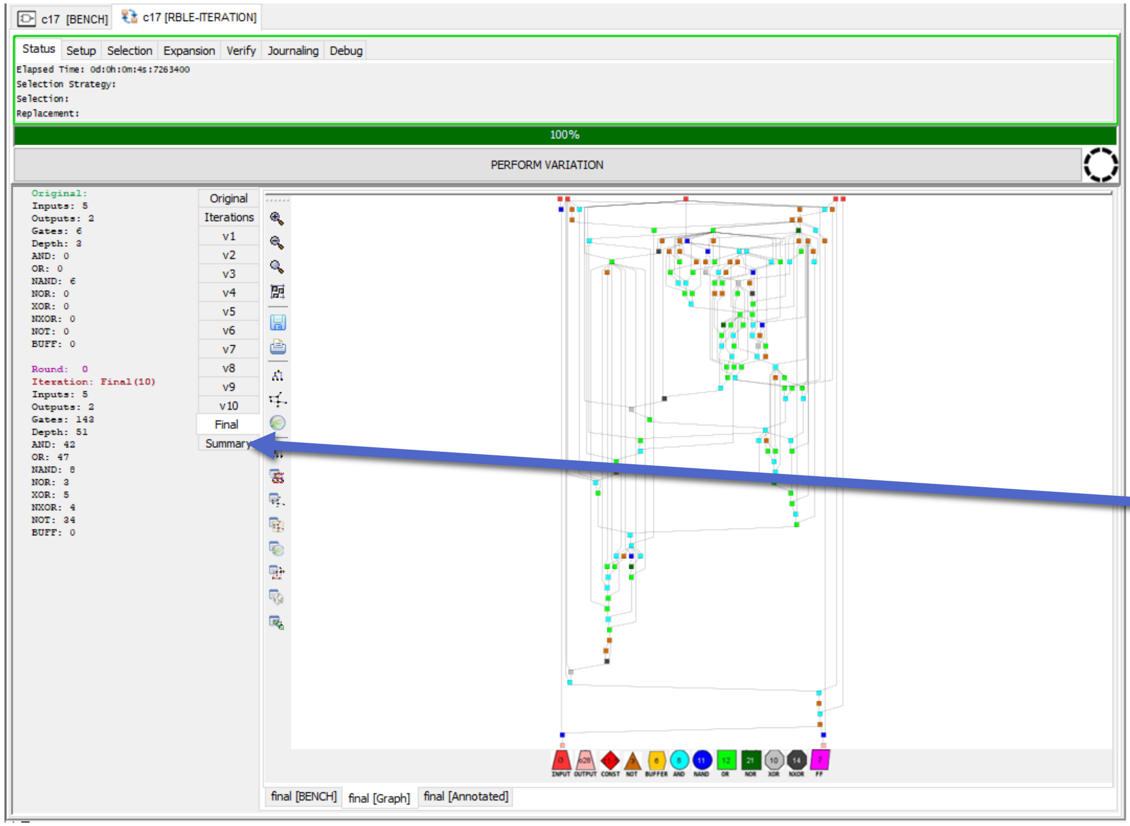
c17 [BENCH] | c17 [Graph] | c17 [IV]

Running an RBLE Experiment

Fill in:

- Setup/Experiment Directory
- Selection/min and max size
- Expansion/Algorithm Type and option value
- Verify and Journal options

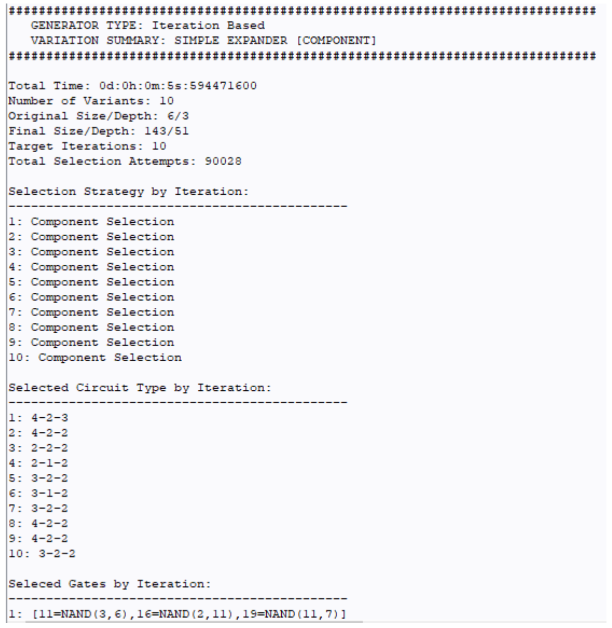
Click PERFORM VARIATION



```
Elapsed Time: 0d:0h:0m:4s:7263400
Selection Strategy:
Selection:
Replacement:
```

```
Original:
Inputs: 5
Outputs: 2
Gates: 6
Depth: 3
AND: 0
OR: 0
NAND: 6
NOR: 0
XOR: 0
NXOR: 0
NOT: 0
BUFF: 0

Round:  0
Iteration: Final(10)
Inputs: 5
Outputs: 2
Gates: 143
Depth: 51
AND: 42
OR: 47
NAND: 8
NOR: 3
XOR: 5
NXOR: 4
NOT: 34
BUFF: 0
```

Summary stats reported at completion



```
################################################################################
   GENERATOR TYPE: Iteration Based
   VARIATION SUMMARY: SIMPLE EXPANDER [COMPONENT]
################################################################################

Total Time: 0d:0h:0m:5s:594471600
Number of Variants: 10
Original Size/Depth: 6/3
Final Size/Depth: 143/51
Target Iterations: 10
Total Selection Attempts: 90028

Selection Strategy by Iteration:
---------------------------------------------
1: Component Selection
2: Component Selection
3: Component Selection
4: Component Selection
5: Component Selection
6: Component Selection
7: Component Selection
8: Component Selection
9: Component Selection
10: Component Selection

Selected Circuit Type by Iteration:
---------------------------------------------
1: 4-2-3
2: 4-2-2
3: 2-2-2
4: 2-1-2
5: 3-2-2
6: 3-1-2
7: 3-2-2
8: 4-2-2
9: 4-2-2
10: 3-2-2

Seleced Gates by Iteration:
---------------------------------------------
1: [11=NAND(3,6),16=NAND(2,11),19=NAND(11,7)]
```

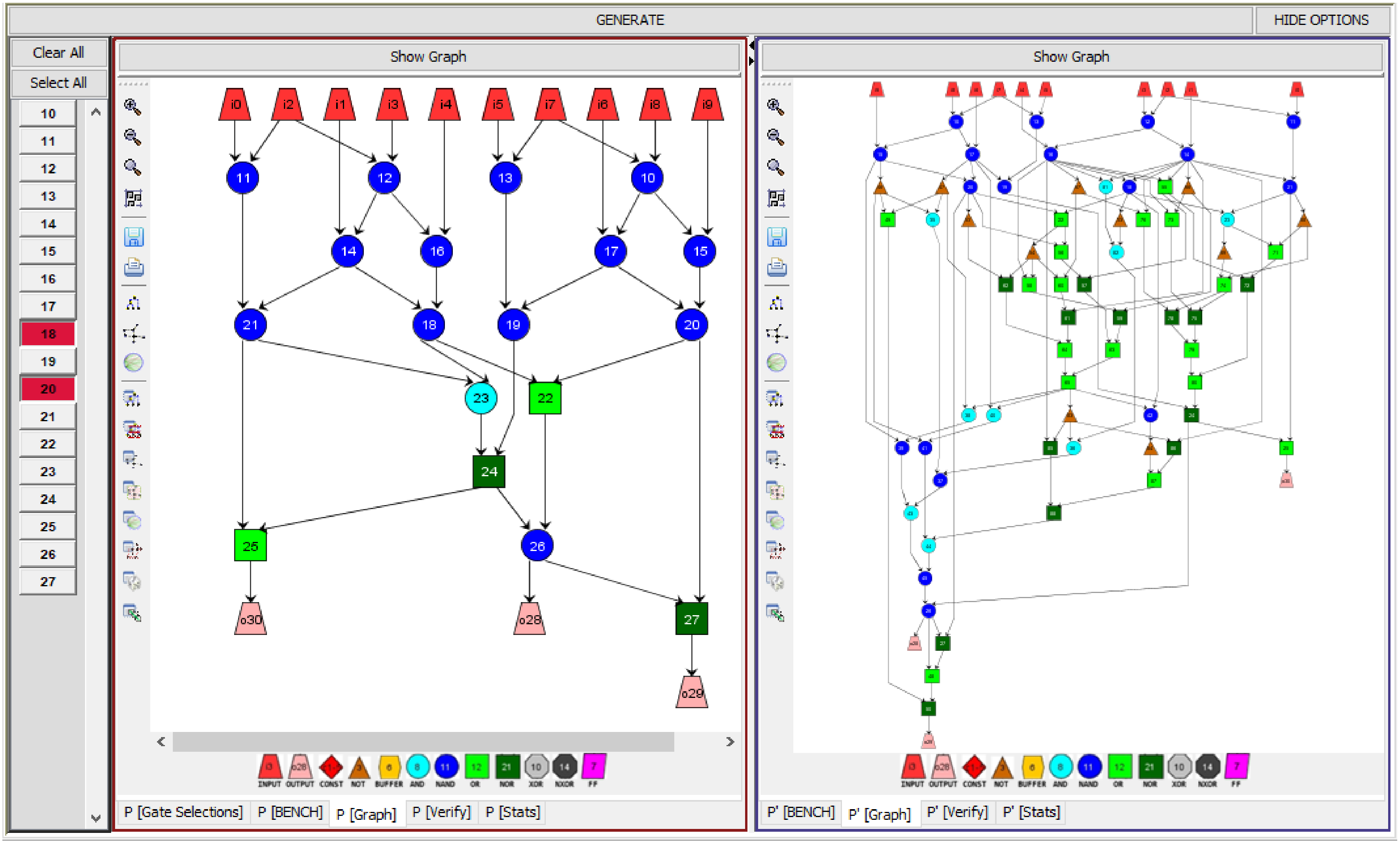
Deterministic Variation

- Deterministic algorithms apply a prescribed set of steps with some elements of pseudo-random choices to achieve a specific goal
- Goal of these algorithms are towards **component hiding**
  - **Defeat algorithms that target semantic identification of components**
- Understanding these algorithms is best done through review of publications where key aspects and experimental results of the algorithms have been disseminated

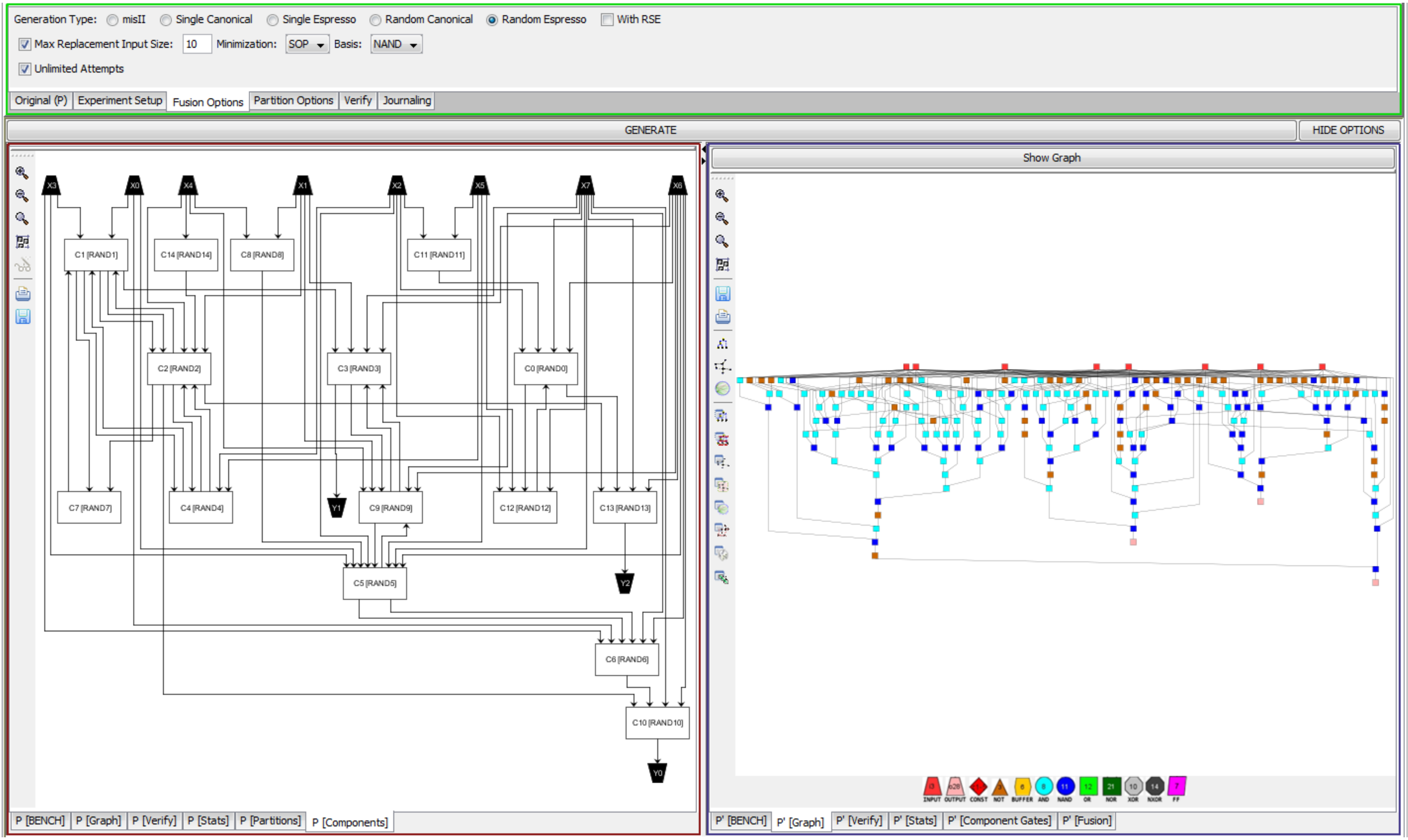
Deterministic Variation

- Boundary Blurring
  - Mutates the type of a gate randomly (for example, from AND to XOR) to change the expected signature at a component boundary
  - Recovery logic is introduced to preserve the original semantics of the gate signal to other gates that depend on it
- Component Fusion
  - Circuit must be partitioned into subcircuit “components”
  - In order to work correctly, components must be defined so that original component boundaries are extended
  - Black-box synthesis is performed creating a virtual black box of each new component
- Component Encryption
  - Similar to component fusion: circuit must be partitioned into a set of component subcircuits
  - Boundaries of components are encoded and decoded internally
  - An implementation of white-box cryptography, where the circuit is replaced with an internal network of encoded look-up-tables

Boundary Blurring



Component Fusion

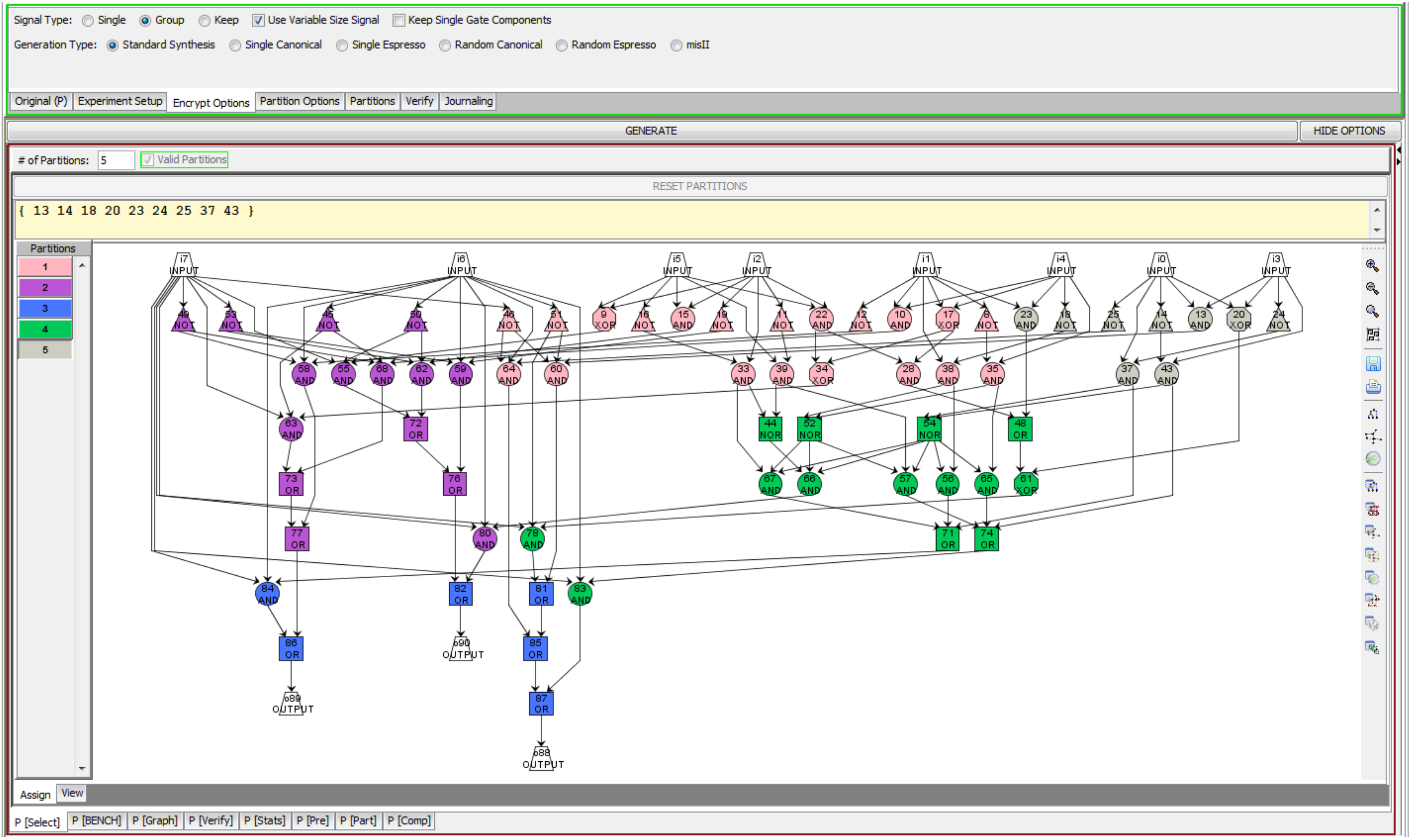


Generation Type: ( ) misII ( ) Single Canonical ( ) Single Espresso ( ) Random Canonical [✓ Random Espresso] ☐ With RSE

☑ Max Replacement Input Size: 10 Minimization: SOP Basis: NAND

☑ Unlimited Attempts

Component Encryption



Signal Type: ( ) Single [✓ Group] ( ) Keep [✓ Use Variable Size Signal] [ ] Keep Single Gate Components

Generation Type: [✓ Standard Synthesis] ( ) Single Canonical ( ) Single Espresso ( ) Random Canonical ( ) Random Espresso ( ) misII

Original (P) | Experiment Setup | Encrypt Options | Partition Options | Partitions | Verify | Journaling

GENERATE | HIDE OPTIONS

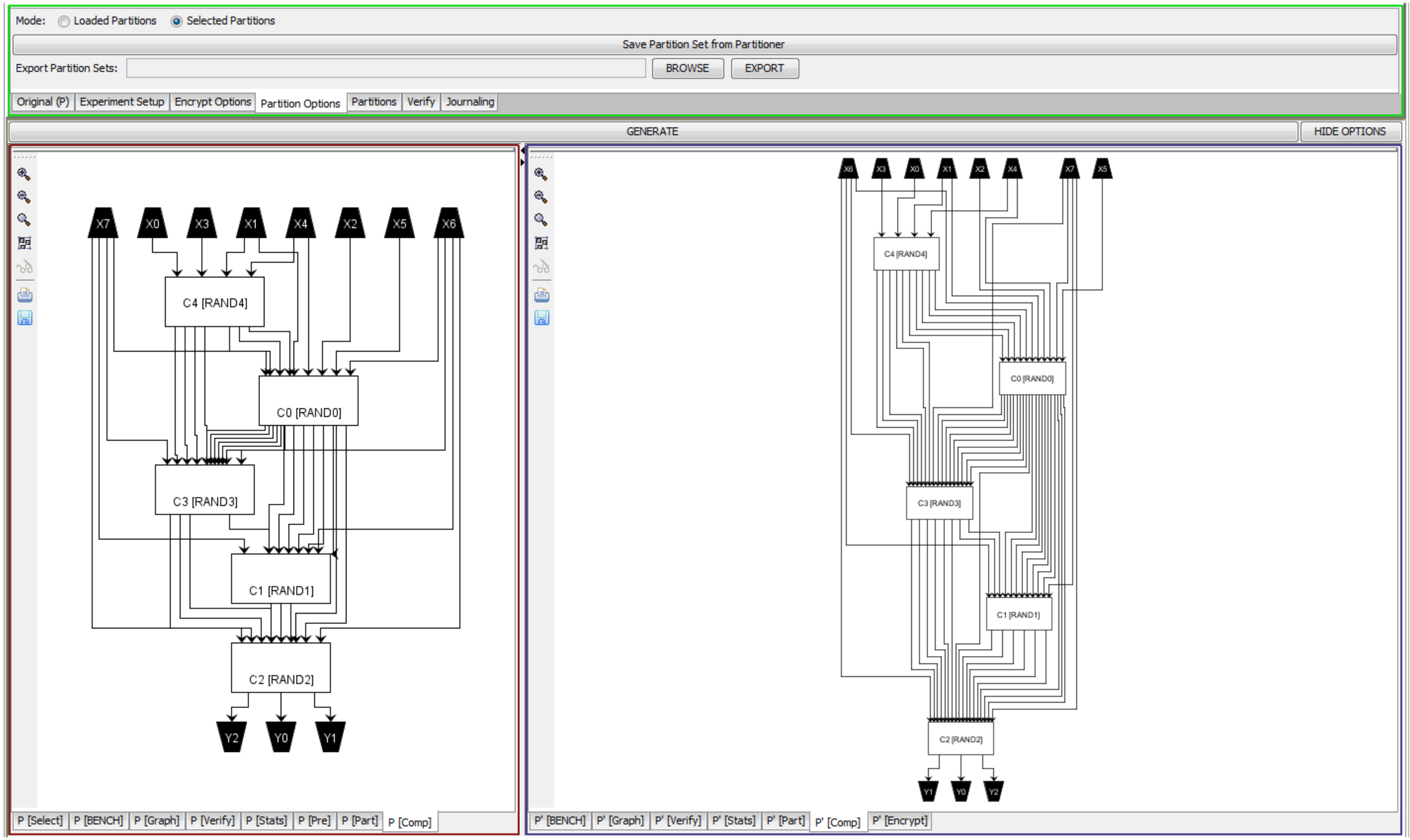
# of Partitions: 5 [✓ Valid Partitions]

RESET PARTITIONS

```
{ 13 14 18 20 23 24 25 37 43 }
```

Partitions
1
2
3
4
5

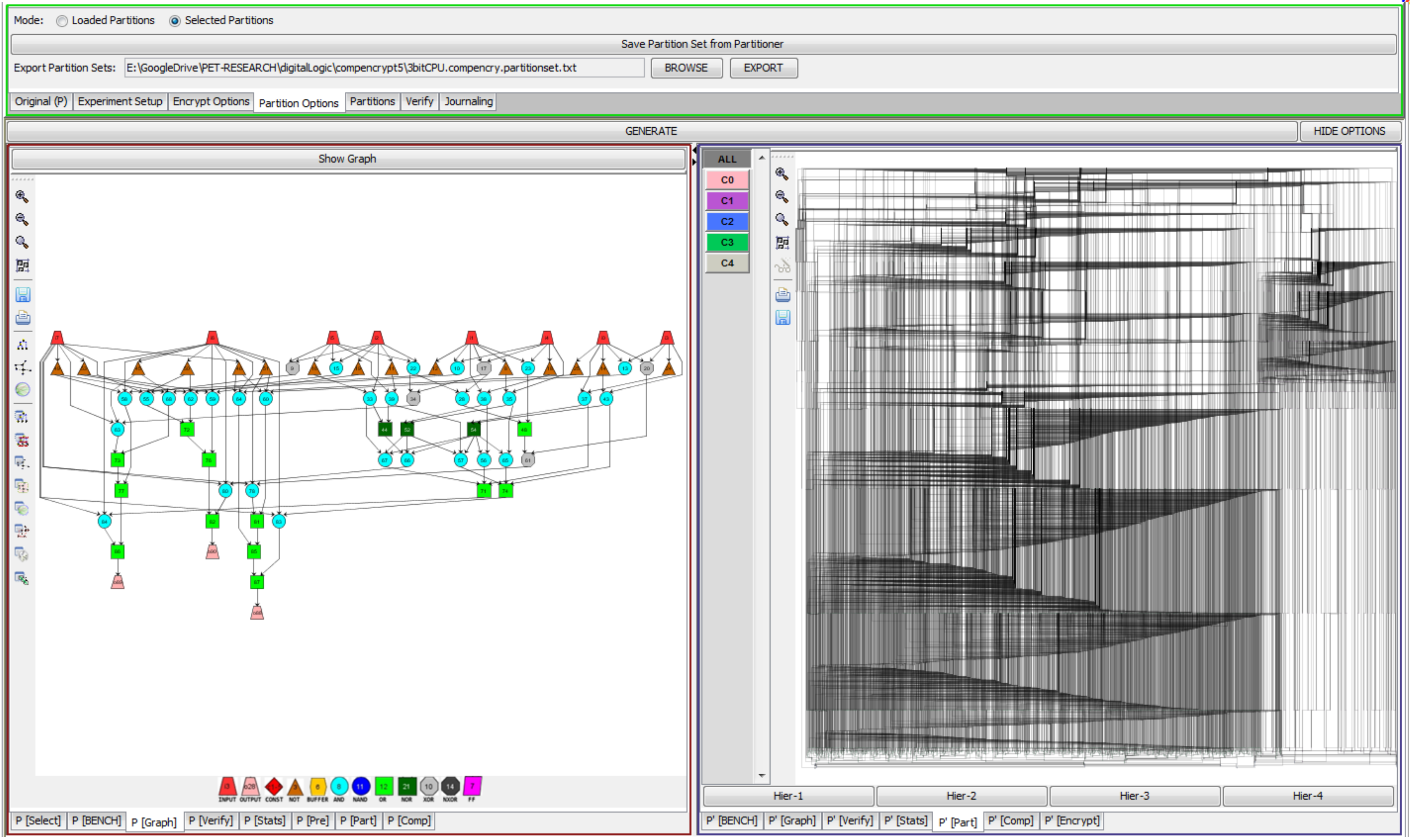
Component Encryption


Mode: Loaded Partitions [✓ Selected Partitions]

Save Partition Set from Partitioner

Export Partition Sets:

Component Encryption

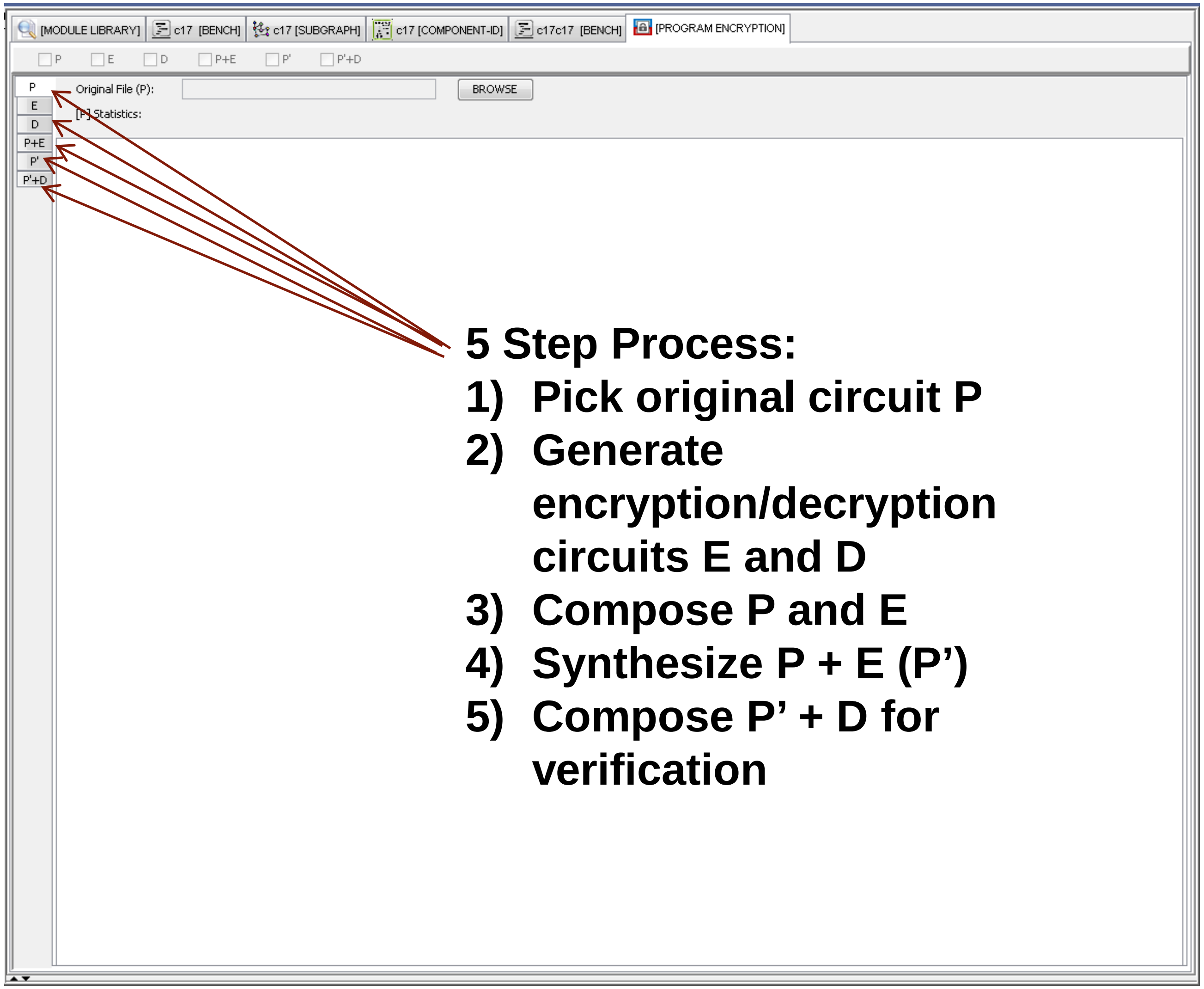


Mode: Loaded Partitions [✓ Selected Partitions]

Export Partition Sets: E:\GoogleDrive\PET-RESEARCH\digitalLogic\compencrypt5\3bitCPU.compencry.partitionset.txt

Program Encryption

This option does not require a selected BENCH circuit to be loaded first

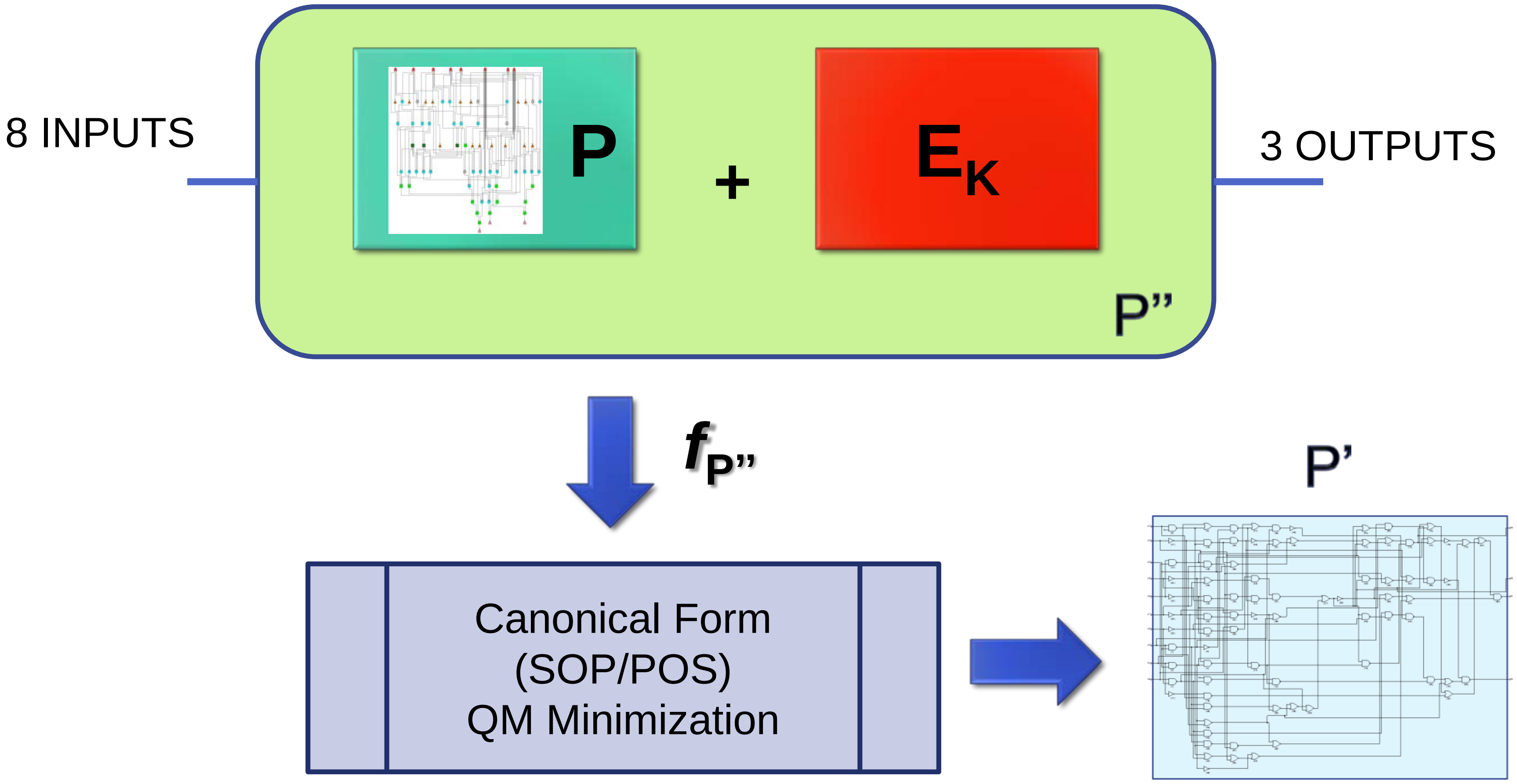


5 Step Process:

1) **Pick original circuit P**
2) **Generate encryption/decryption circuits E and D**
3) **Compose P and E**
4) **Synthesize P + E (P')**
5) **Compose P' + D for verification**

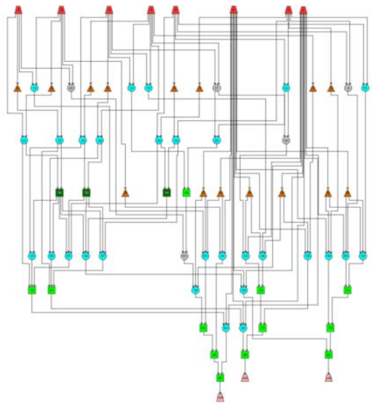
Program Encryption

- Basic Overview



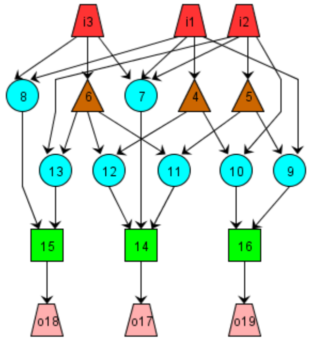
Summary of Program Encryption

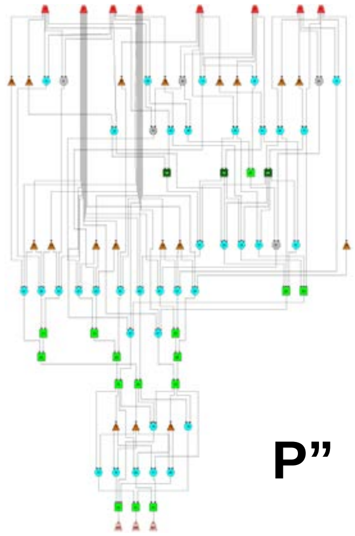
P



+

E_k



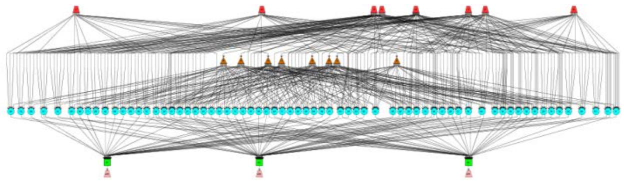


P''

$f_{P''}$

```
00000000|888
01234567|567
------------
00000000|100
00000001|100
00000010|110
00000011|000
00000100|100
00000101|000
00000110|110
00000111|101
00001000|100
00001001|111
............
............
11111100|110
11111101|010
11111110|100
11111111|000
```

Canonical Form
(SOP/POS)
QM Minimization

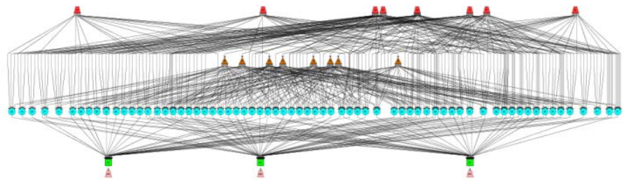


P'

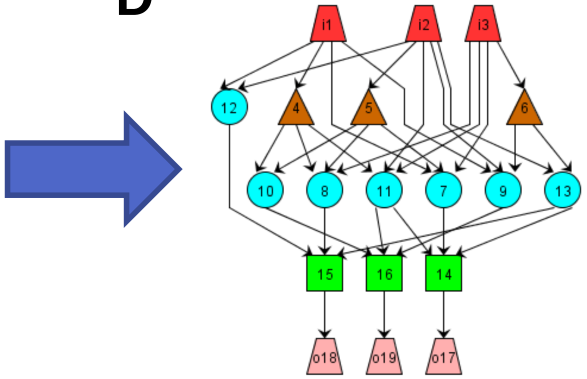
Summary of Program Encryption

- **A legitimate user of P' can reproduce the functionality of P with the decryption circuit D:**

P'



D



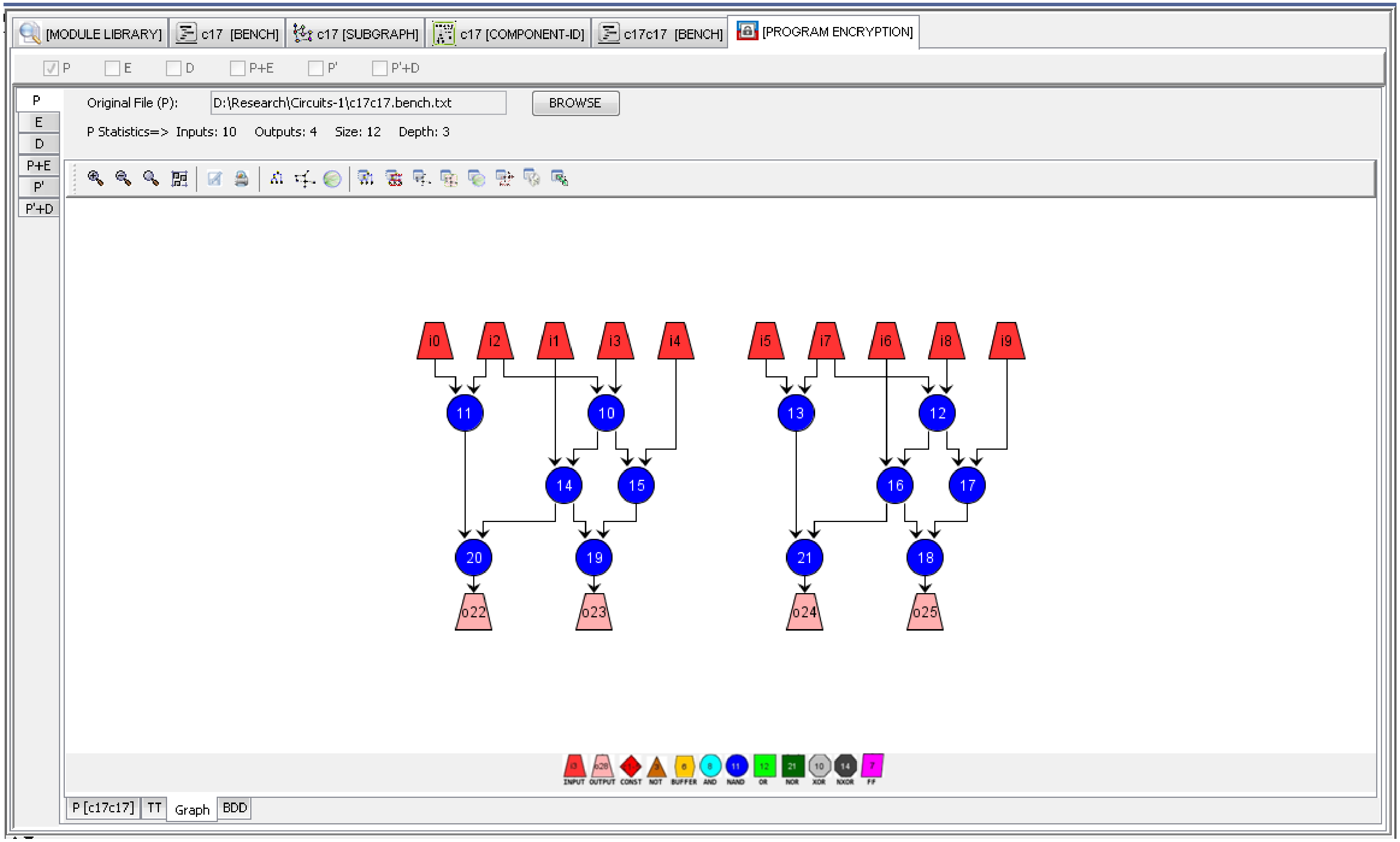
f_P

- Some things about P'
  - There is no seam (P + E)
  - There are no components related to P or E
  - There is no topology related to P or E
  - There is nothing directly tied to the structure/configuration of P or E

  - **There is ONLY the information necessary to compute:**

$$P'(x) = E_K(P(x)), \forall x$$

Program Encryption: Choose P

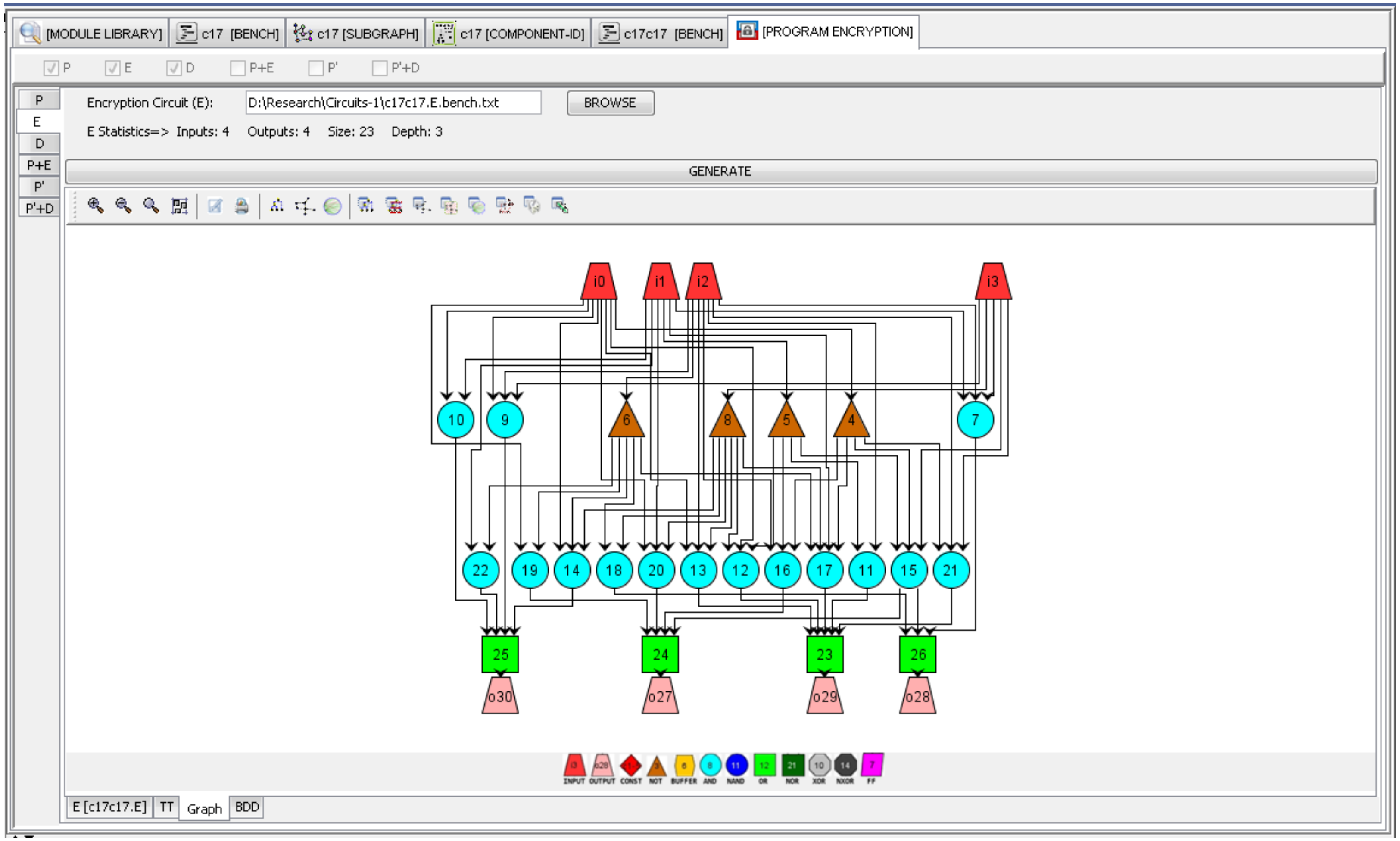

[MODULE LIBRARY] | c17 [BENCH] | c17 [SUBGRAPH] | c17 [COMPONENT-ID] | c17c17 [BENCH] | [PROGRAM ENCRYPTION]

[✓ P] E D P+E P' P'+D

Original File (P): D:\Research\Circuits-1\c17c17.bench.txt BROWSE

P Statistics=> Inputs: 10 Outputs: 4 Size: 12 Depth: 3

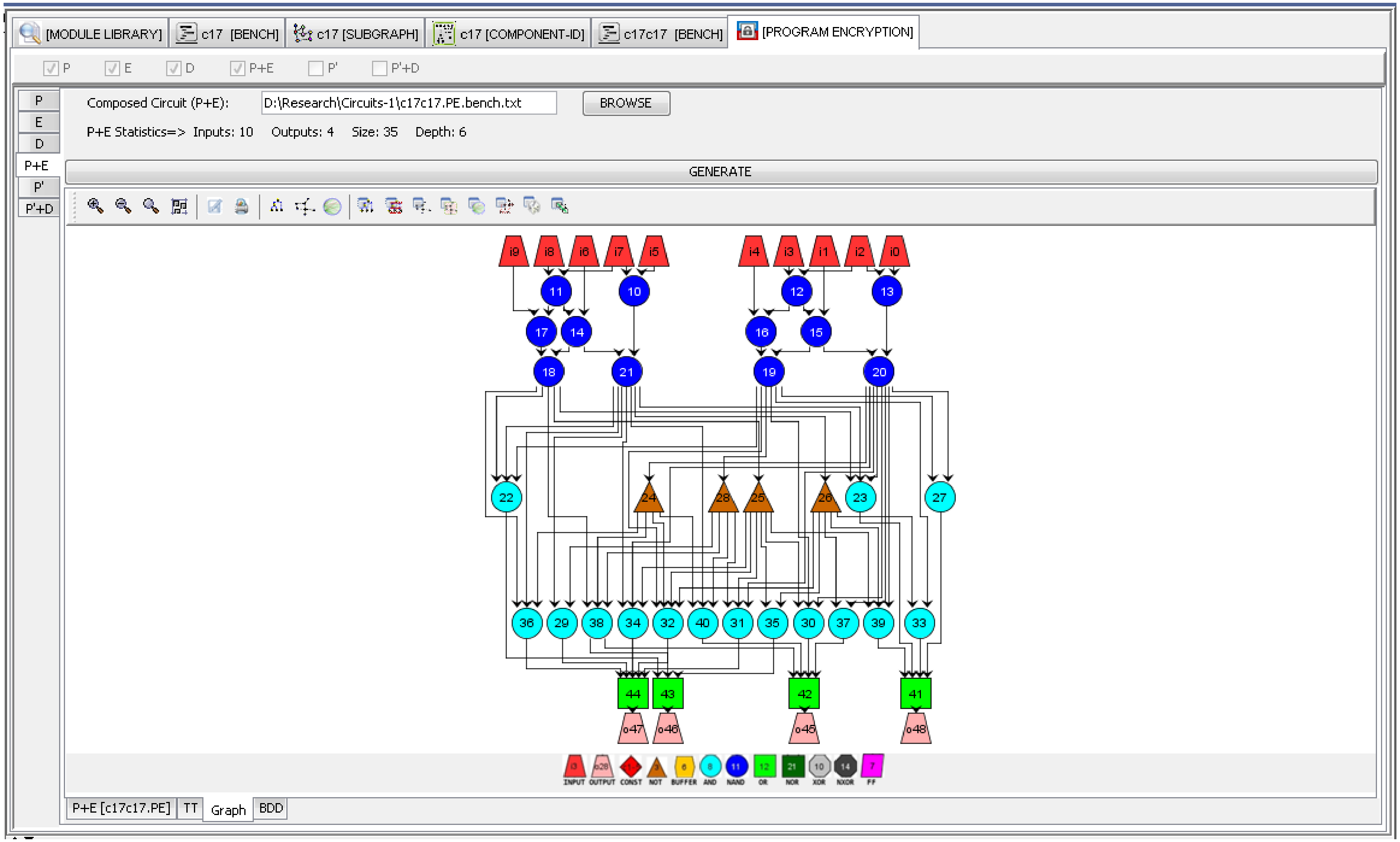
Program Encryption: Generate E and D


[MODULE LIBRARY] | c17 [BENCH] | c17 [SUBGRAPH] | c17 [COMPONENT-ID] | c17c17 [BENCH] | [PROGRAM ENCRYPTION]

Encryption Circuit (E): D:\Research\Circuits-1\c17c17.E.bench.txt

E Statistics=> Inputs: 4 Outputs: 4 Size: 23 Depth: 3

Program Encryption: Compose P and E



[MODULE LIBRARY] | c17 [BENCH] | c17 [SUBGRAPH] | c17 [COMPONENT-ID] | c17c17 [BENCH] | [PROGRAM ENCRYPTION]

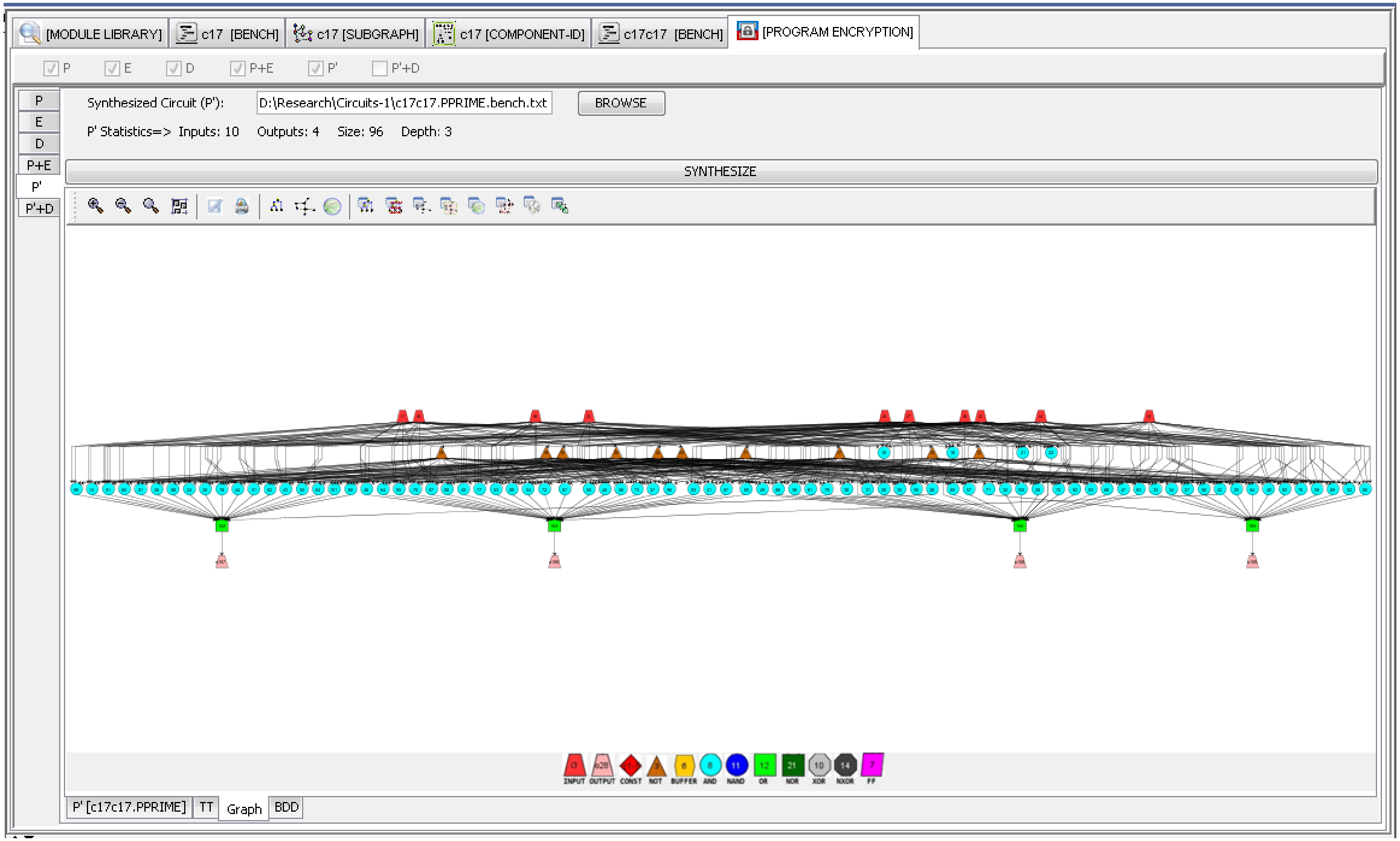
☑ P ☑ E ☑ D ☑ P+E ☐ P' ☐ P'+D

Composed Circuit (P+E): D:\Research\Circuits-1\c17c17.PE.bench.txt BROWSE

P+E Statistics=> Inputs: 10 Outputs: 4 Size: 35 Depth: 6

GENERATE

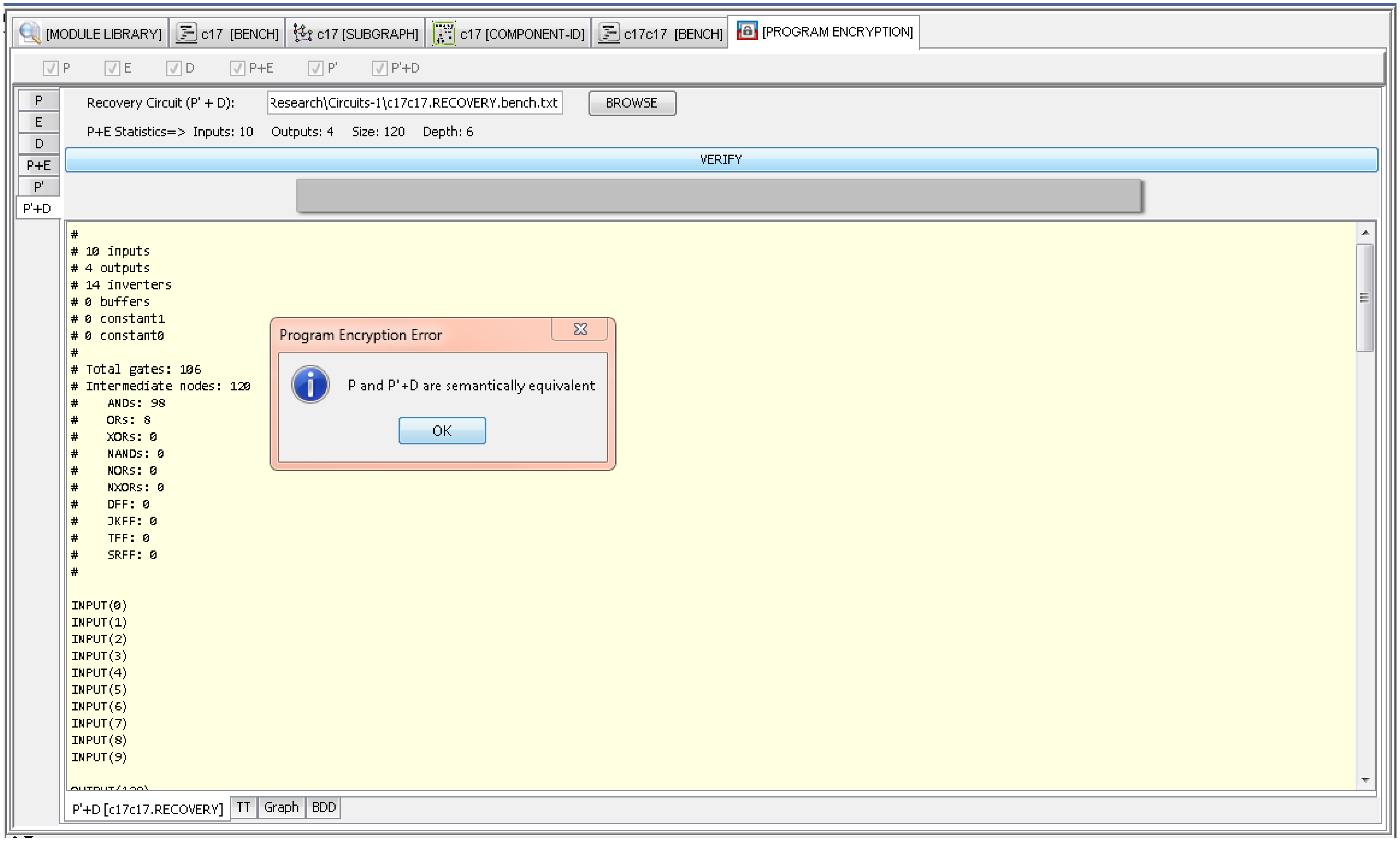
Program Encryption: Synthesize P ‘



Synthesized Circuit (P'): D:\Research\Circuits-1\c17c17.PPRIME.bench.txt

P' Statistics=> Inputs: 10 Outputs: 4 Size: 96 Depth: 3

Program Encryption: Concat D to P' for Verification



Recovery Circuit (P' + D): Research\Circuits-1\c17c17.RECOVERY.bench.txt

P+E Statistics=> Inputs: 10 Outputs: 4 Size: 120 Depth: 6

```
#
# 10 inputs
# 4 outputs
# 14 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 106
# Intermediate nodes: 120
#    ANDs: 98
#    ORs: 8
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
#    JKFF: 0
#    TFF: 0
#    SRFF: 0
#

INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)
```

Program Encryption Error

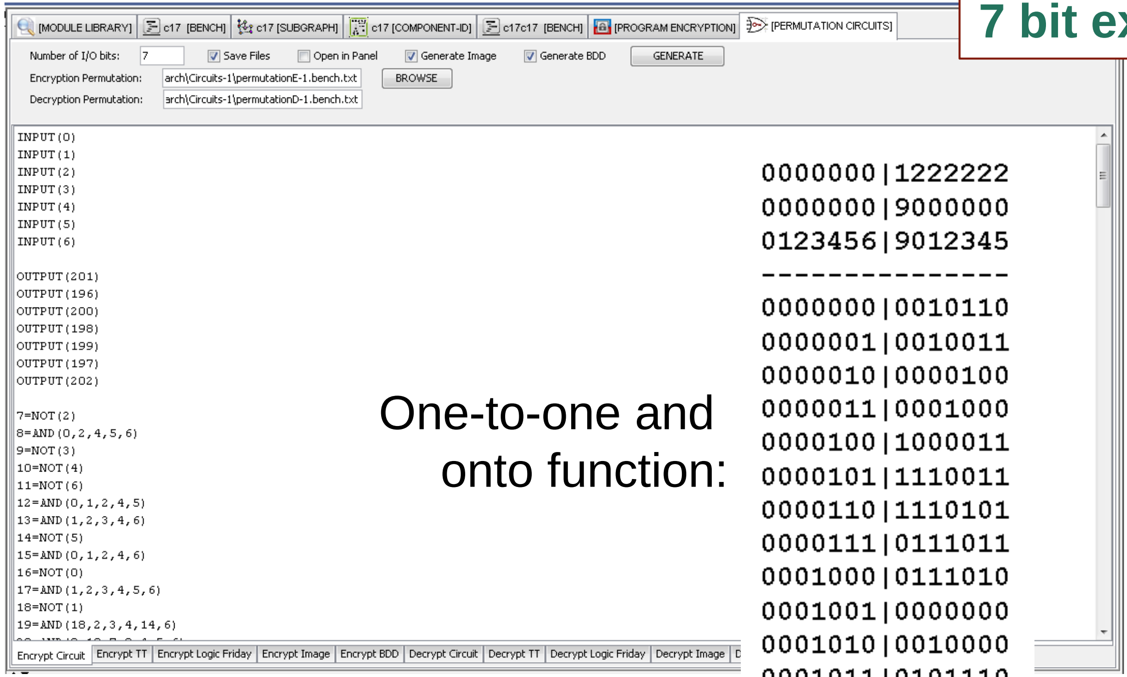
P and P'+D are semantically equivalent

Permutation Circuits

This option does not require a selected BENCH circuit to be loaded first

This generates permutation circuits (E), with a corresponding decryption circuit (D)

7 bit example



Number of I/O bits: 7 [✓ Save Files] Open in Panel [✓ Generate Image] [✓ Generate BDD]

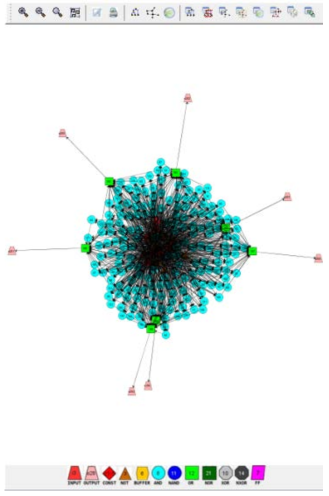
Encryption Permutation: arch\Circuits-1\permutationE-1.bench.txt

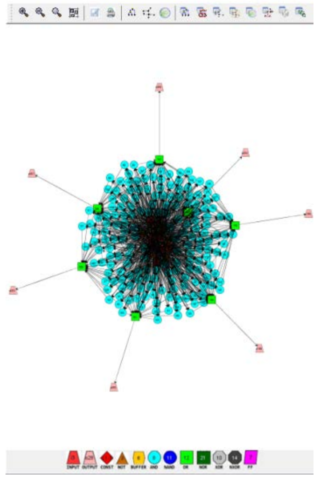
Decryption Permutation: arch\Circuits-1\permutationD-1.bench.txt

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)

OUTPUT(201)
OUTPUT(196)
OUTPUT(200)
OUTPUT(198)
OUTPUT(199)
OUTPUT(197)
OUTPUT(202)

7=NOT(2)
8=AND(0,2,4,5,6)
9=NOT(3)
10=NOT(4)
11=NOT(6)
12=AND(0,1,2,4,5)
13=AND(1,2,3,4,6)
14=NOT(5)
15=AND(0,1,2,4,6)
16=NOT(0)
17=AND(1,2,3,4,5,6)
18=NOT(1)
19=AND(18,2,3,4,14,6)
```

One-to-one and onto function:

```
0000000|1222222
0000000|9000000
0123456|9012345
---------------
0000000|0010110
0000001|0010011
0000010|0000100
0000011|0001000
0000100|1000011
0000101|1110011
0000110|1110101
0000111|0111011
0001000|0111010
0001001|0000000
0001010|0010000
0001011|0101110
0001100|0100101
0001101|1011110
0001110|0001100
0001111|1100110
0010000|0101011
```

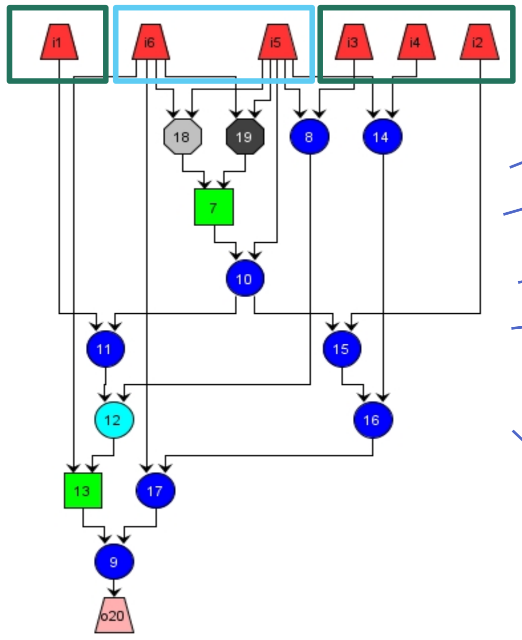

E


D

Functional Polymorphism

- **Functional Polymorphism**
  - **ONE FORM, MANY FUNCTIONS**

P



XOR

1	2	3
0	0	0
0	1	1
1	0	1
1	1	0

NAND

1	2	3
0	0	1
0	1	1
1	0	1
1	1	1

AND

1	2	3
0	0	0
0	1	0
1	0	0
1	1	1

CONST1

1	2	3
0	0	1
0	1	1
1	0	1
1	1	1

OR

1	2	3
0	0	0
0	1	1
1	0	1
1	1	1

NXOR

1	2	3
0	0	1
0	1	0
1	0	0
1	1	1

NOR

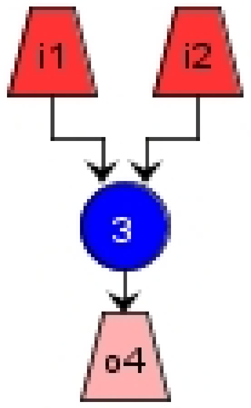
1	2	3
0	0	1
0	1	0
1	0	0
1	1	0

This one circuit could produce an AND, OR, XOR, NXOR, NOR, NAND or OTHER functions

What is a Polygate?

NAND Gate

```
12|4
----
00|1
01|1
10|1
11|0
```



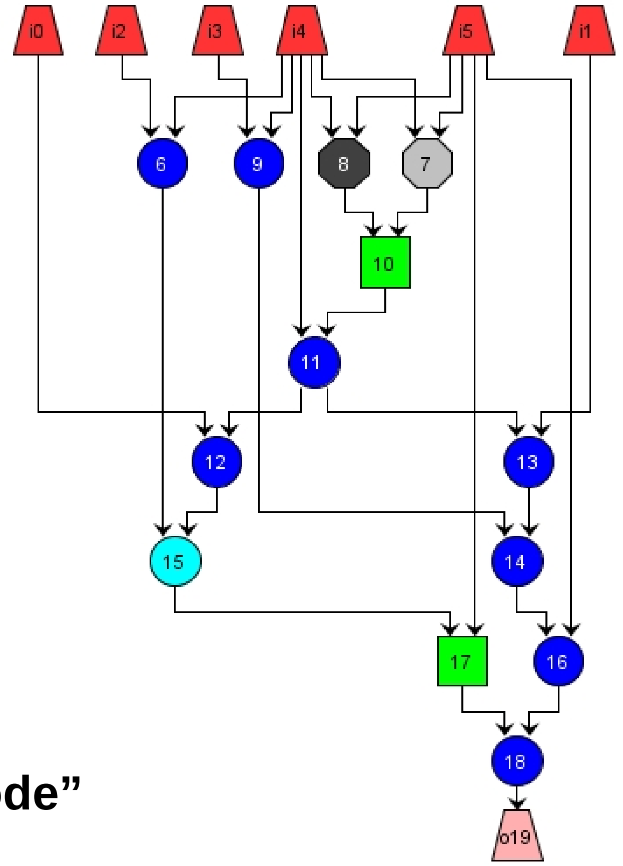
Key Bits

Original Inputs

```
000000|1
012345|9
---------
111000|1
111001|1
111010|1
111011|0
```

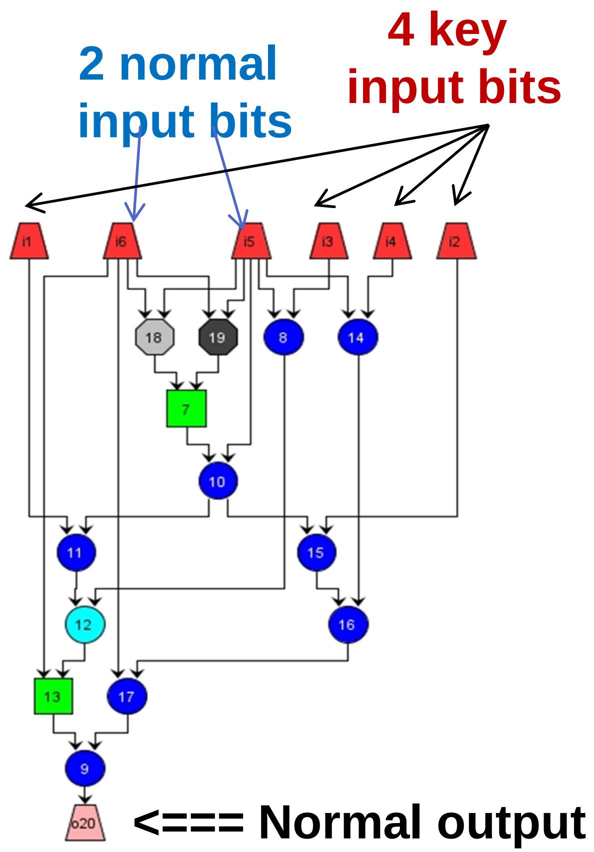
"MSB Mode"

Polygate Version



What is a Polygate?

- In circuits: realizable as a “polymorphic gate” or “polygate”



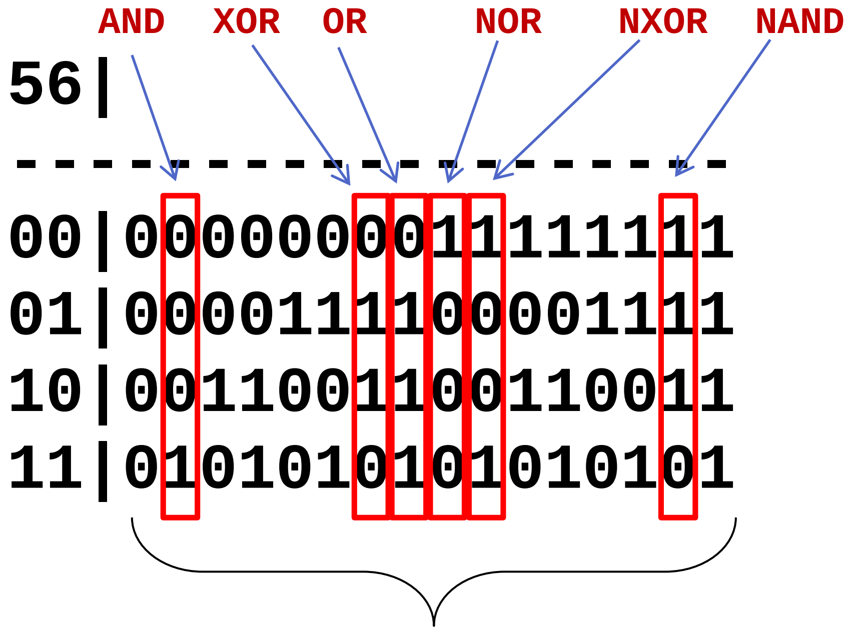
Essentially a BINARY (2-input) GATE

4 additional inputs CHOOSE the function

The 4 inputs form a “functional key”

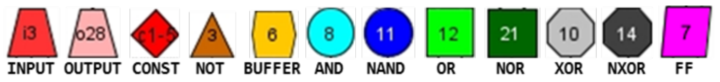
Key must be provided to get correct function

This form can generate 16 functions on 2 inputs



```
        AND   XOR  OR      NOR     NXOR   NAND
56|
-----------------------
00|0000000011111111
01|0000111100001111
10|0011001100110011
11|0101010101010101
```

Possible function keys

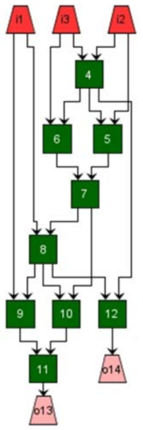


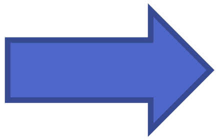
What is a Polygate?

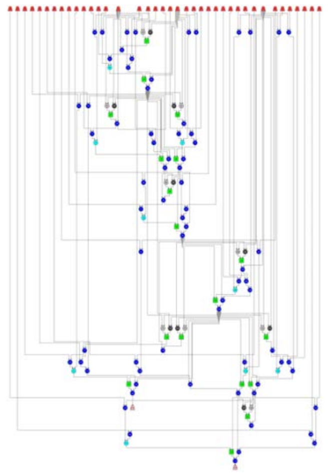
- Every binary gate (2 input/1 output) is transformed to a polygate component (6 input/1 output)
- Appropriate key bits must be provided to the polygate component to execute the correct function
- A polygate is essentially a MULTIPLEXOR component

- General idea:
  - Adversary cannot fully determine circuit function without full I/O enumeration
  - The function of the circuit and therefore components are not know without context of the ey

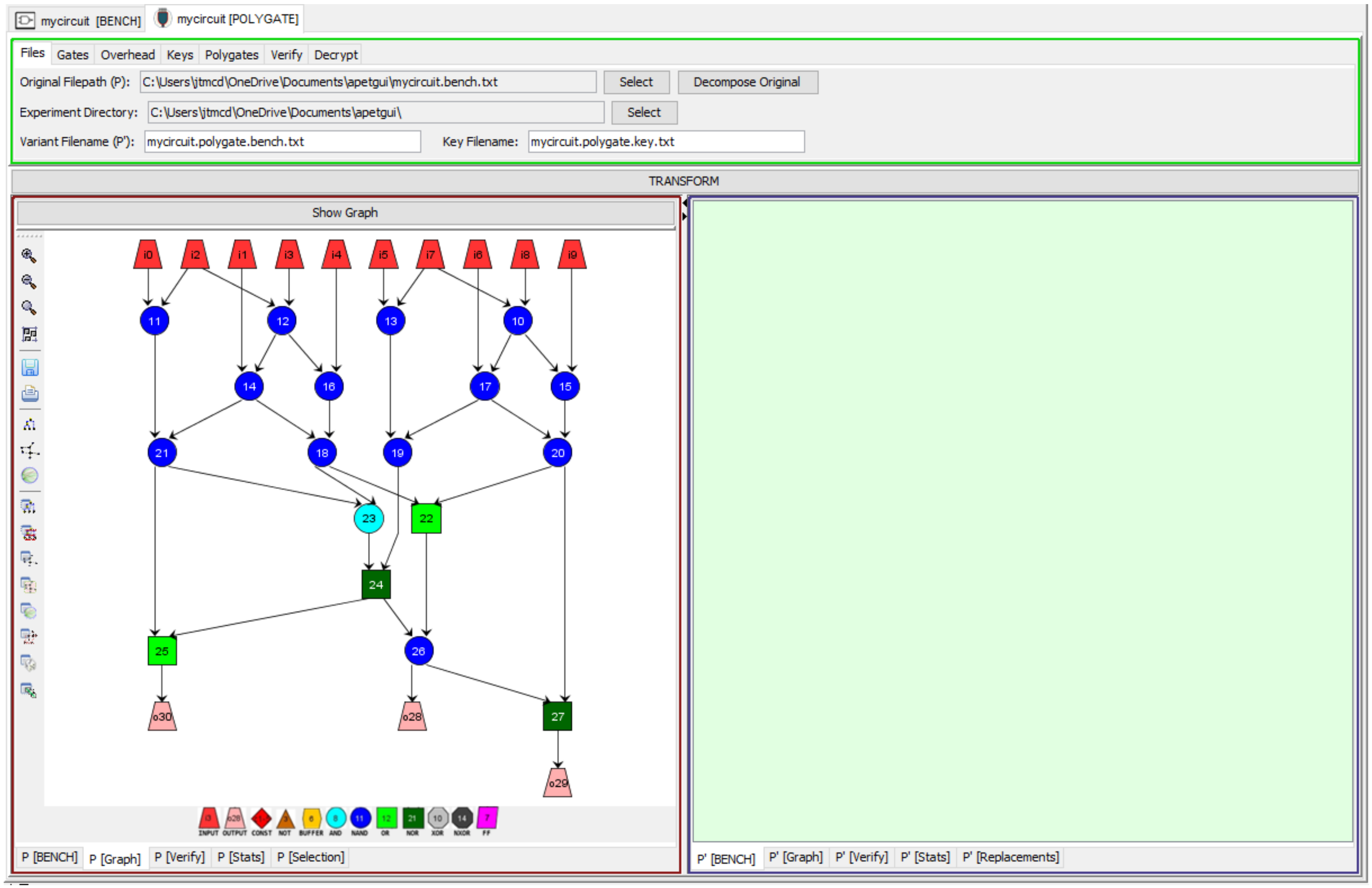
- Overhead:
  - Depends on MUX chosen
  - 4 additional inputs for every binary gate







1) Setup directory and file information



Original Filepath (P): C:\Users\jtmcd\OneDrive\Documents\apetgui\mycircuit.bench.txt

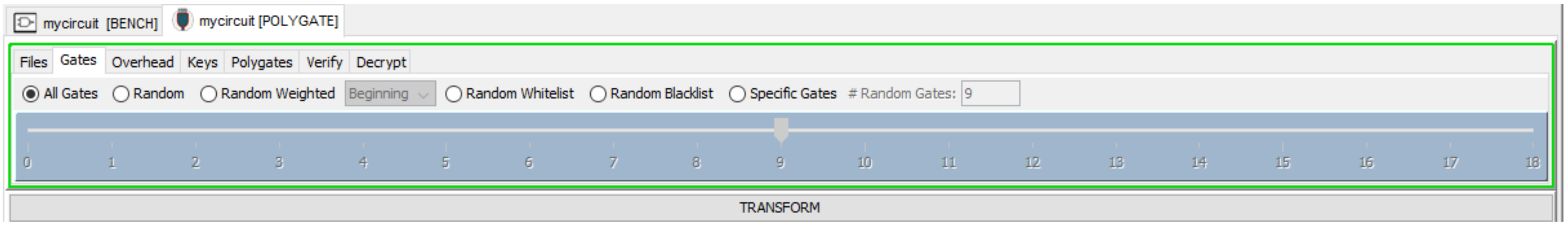
Experiment Directory: C:\Users\jtmcd\OneDrive\Documents\apetgui\

Variant Filename (P'): mycircuit.polygate.bench.txt

Key Filename: mycircuit.polygate.key.txt

2) Pick which gates will be transformed

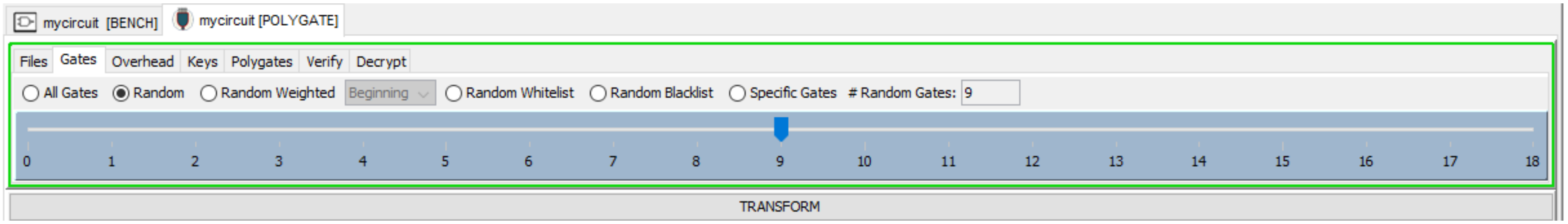
ALL GATES


mycircuit [BENCH] | mycircuit [POLYGATE]

Files | Gates | Overhead | Keys | Polygates | Verify | Decrypt

[✓ All Gates] Random | Random Weighted | Beginning | Random Whitelist | Random Blacklist | Specific Gates | # Random Gates: 9

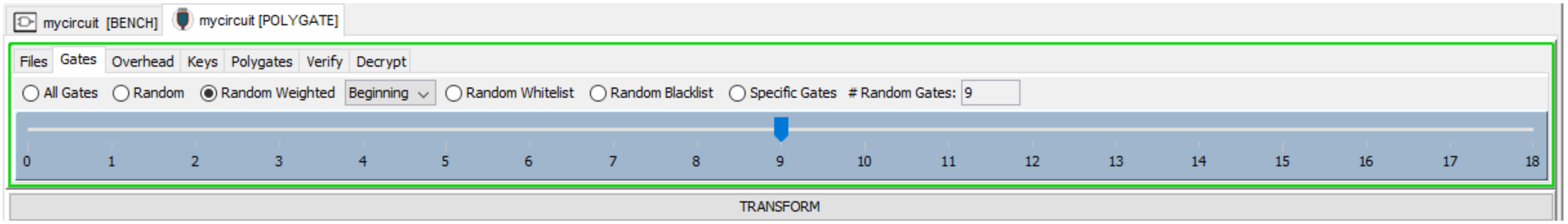
PICK RANDOM # of GATES


mycircuit [BENCH] | mycircuit [POLYGATE]

Files | Gates | Overhead | Keys | Polygates | Verify | Decrypt

All Gates [✓ Random] Random Weighted | Beginning | Random Whitelist | Random Blacklist | Specific Gates | # Random Gates: 9

PICK RANDOM # of GATES, but favor BEGINNING, MIDDLE, or END GATES

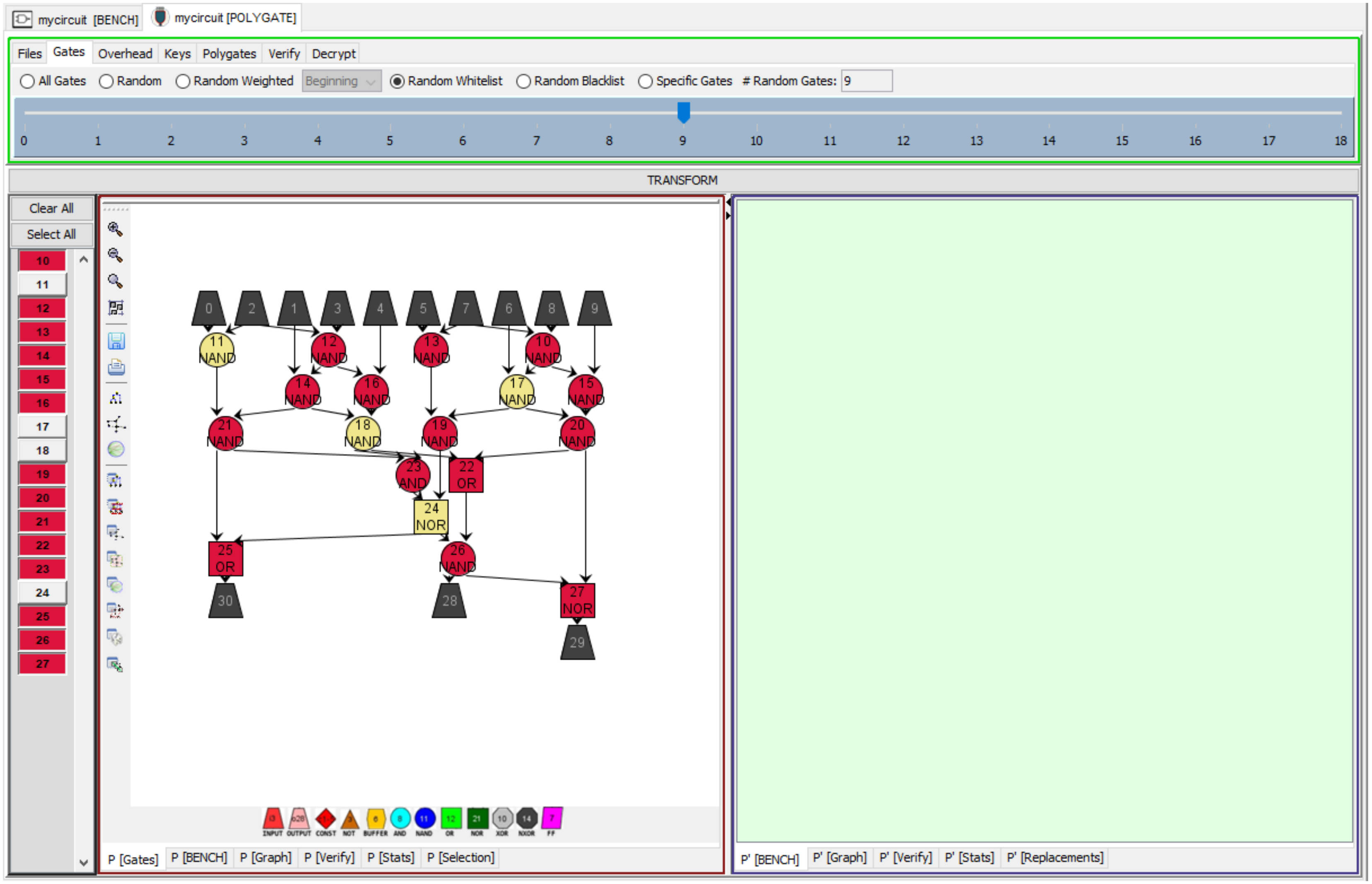

mycircuit [BENCH] | mycircuit [POLYGATE]

Files | Gates | Overhead | Keys | Polygates | Verify | Decrypt

All Gates | Random [✓ Random Weighted] Beginning | Random Whitelist | Random Blacklist | Specific Gates | # Random Gates: 9

2) Pick which gates will be transformed

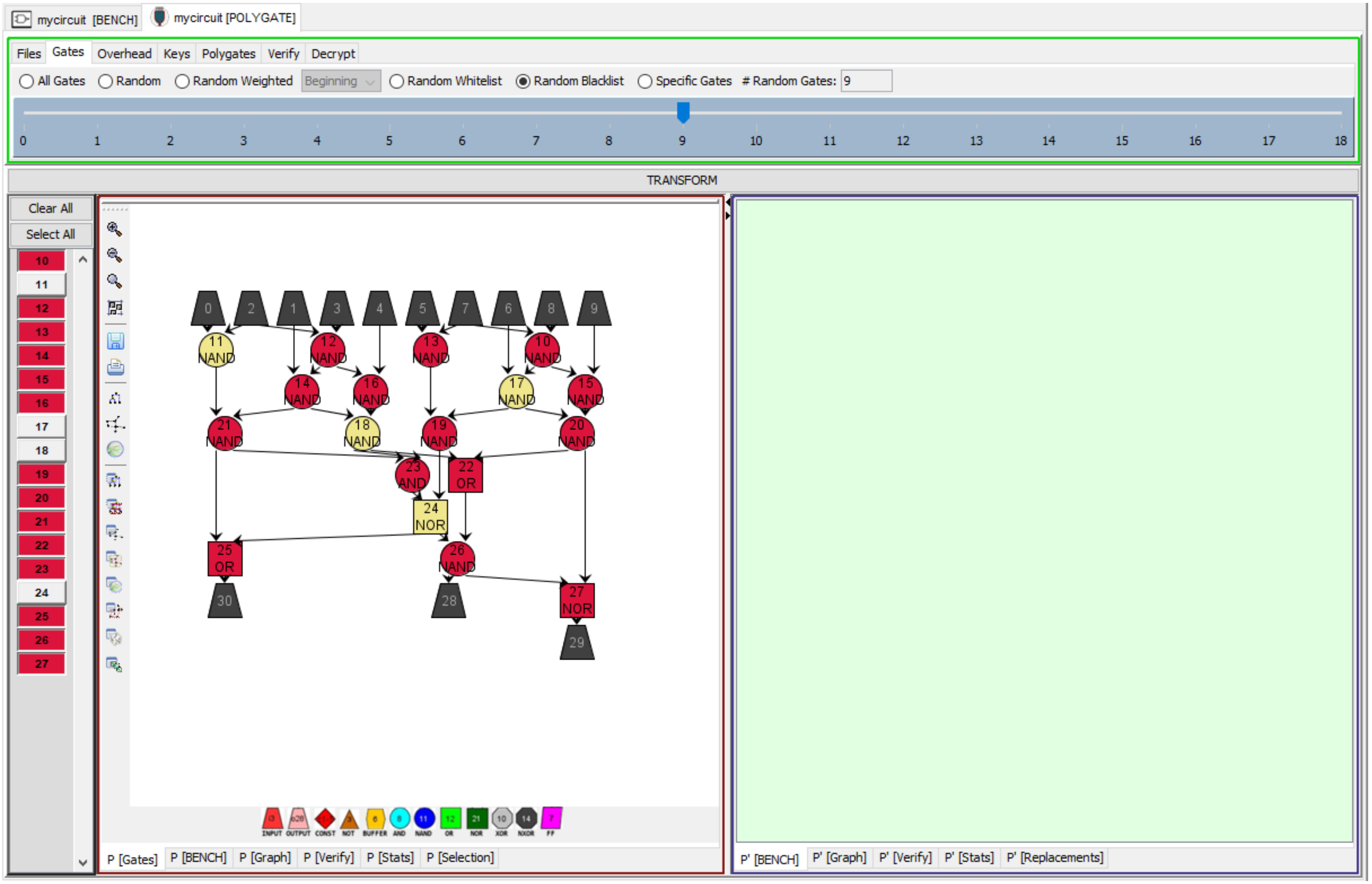
PICK RANDOM # of GATES from a LIST of SELECTED GATES (WHITELIST)



All Gates Random Random Weighted Beginning [✓ Random Whitelist] Random Blacklist Specific Gates # Random Gates: 9

2) Pick which gates will be transformed

PICK RANDOM # of GATES but exclude those on LIST of SELECTED GATES (BLACKLIST)

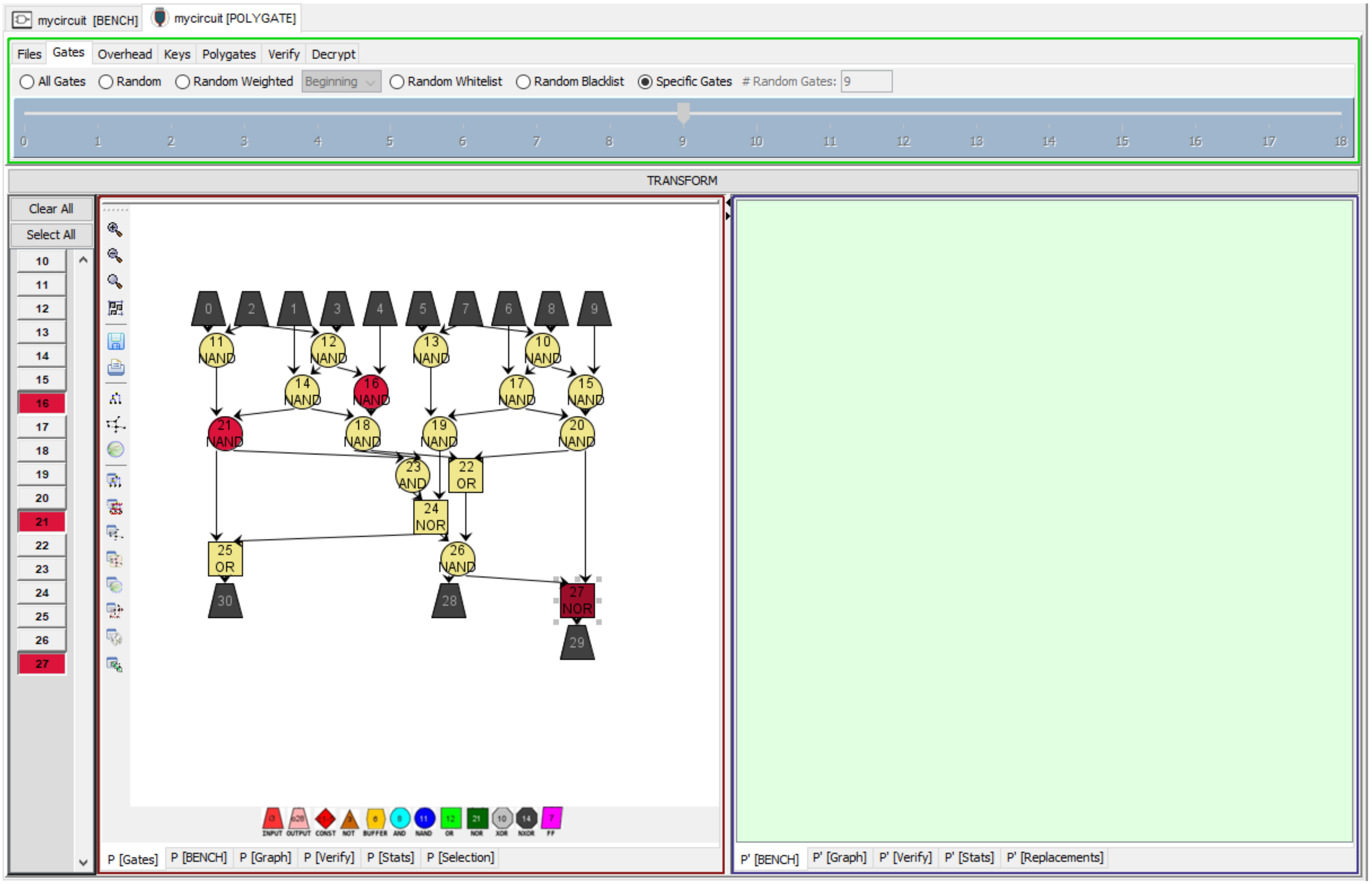


All Gates | Random | Random Weighted | Beginning | Random Whitelist | [✓ Random Blacklist] | Specific Gates | # Random Gates: 9

Polygate Experiments

2) Pick which gates will be transformed

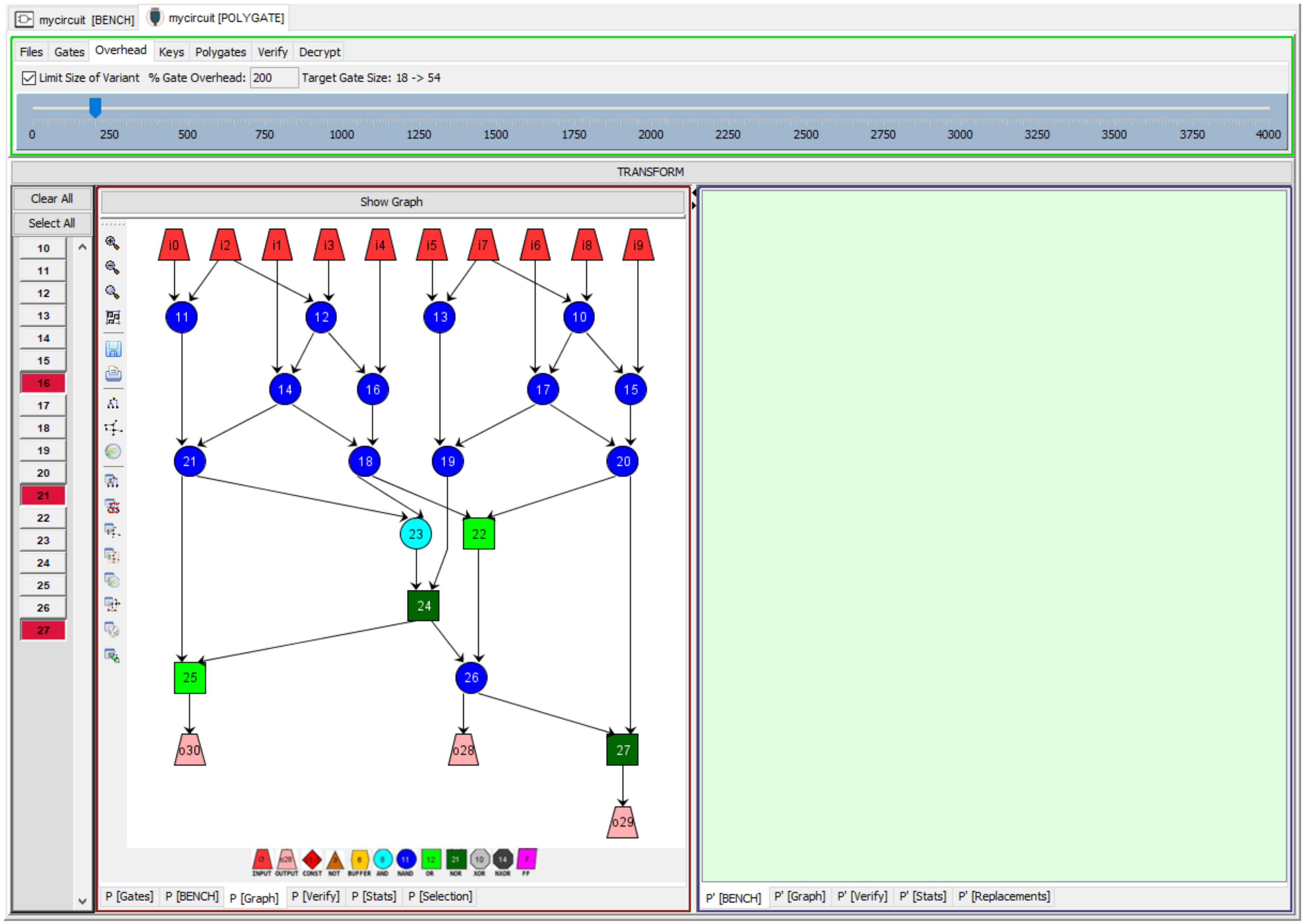
PICK SPECIFIC GATES



( ) All Gates ( ) Random ( ) Random Weighted Beginning ( ) Random Whitelist ( ) Random Blacklist [✓ Specific Gates] # Random Gates: 9

3) Choose overhead limits

Uncheck Limit Size of Variant for no limitations
Otherwise, choose gate % increase allowed (overhead)



mycircuit [BENCH] | mycircuit [POLYGATE]

Files | Gates | Overhead | Keys | Polygates | Verify | Decrypt

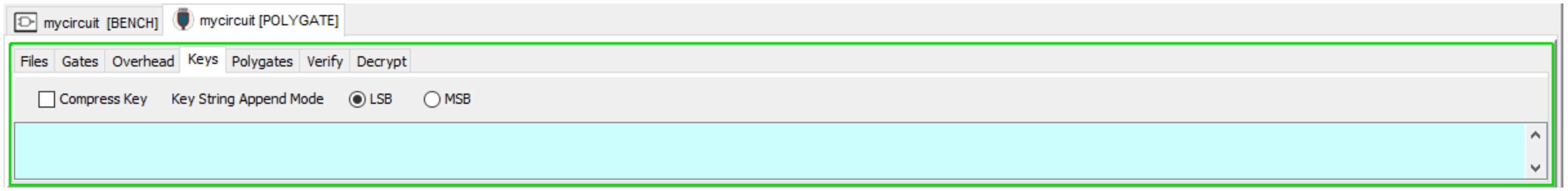
☑ Limit Size of Variant % Gate Overhead: 200 Target Gate Size: 18 -> 54

Polygate Experiments

4) Key options:

Prepend (MSB) or append (LSB) key bits to the input vector

Compress key: reduce the size of the key string (will produce non-deterministic variation)

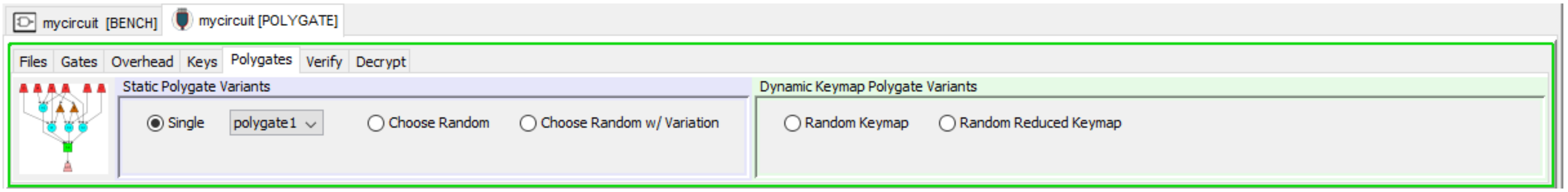


☐ Compress Key   Key String Append Mode   [✓ LSB]   MSB

5) Polygate options:

Static Polygates:
Pick one of the static polygate (MUX) designs: choose specific one (Single)
Pick one of the static versions randomly every time a polygate is inserted (Choose Random)
Pick a random static version and introduce some degree of variation (Random w/ Variation)

Dynamic Keymap:
Generate a random keymap for every polygate (a different truth table mapping for the MUX)
Generate only the 6 basic gate types for the keymap component (Random Reduced)

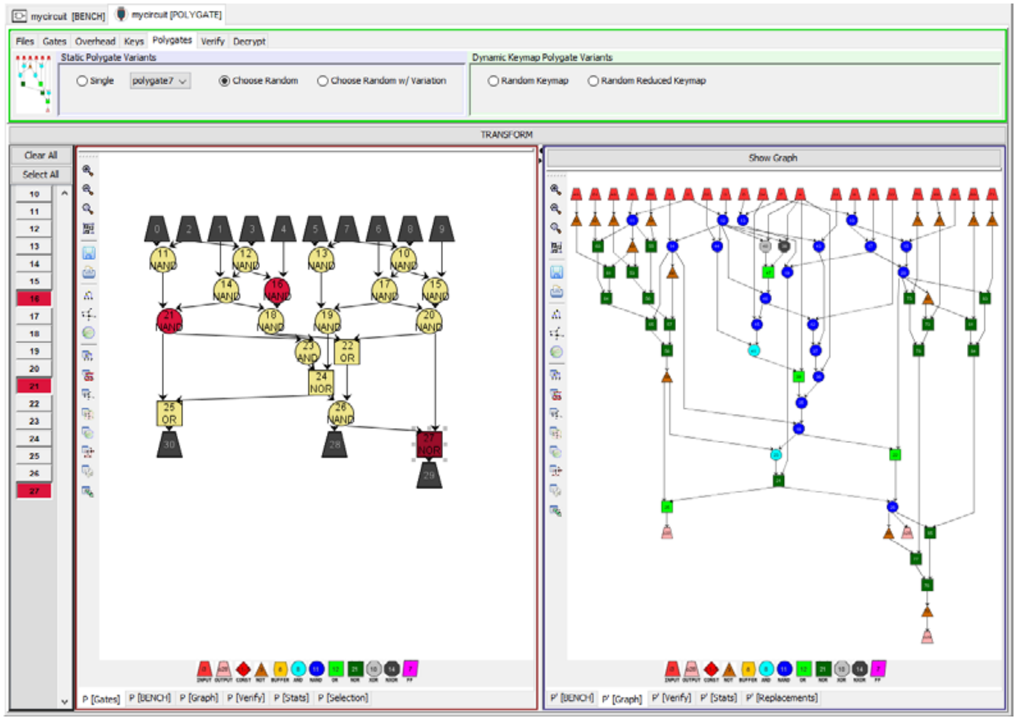


Static Polygate Variants: [✓ Single] polygate1   Choose Random   Choose Random w/ Variation

Dynamic Keymap Polygate Variants: Random Keymap   Random Reduced Keymap

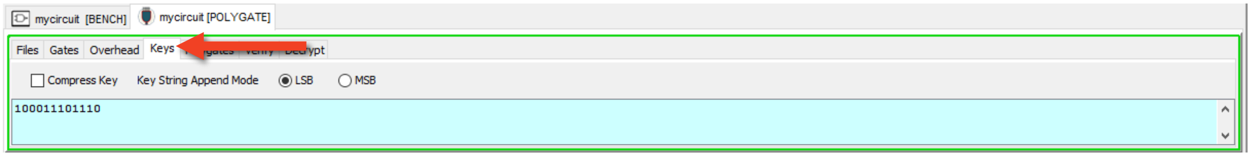
6) Click TRANSFORM

Polygate Experiments


mycircuit [BENCH]
mycircuit [POLYGATE]
Files Gates Overhead Keys Polygates Verify Decrypt
Static Polygate Variants
Single
polygate7
Choose Random
Choose Random w/ Variation
Dynamic Keymap Polygate Variants
Random Keymap
Random Reduced Keymap
TRANSFORM
Clear All
Select All
Show Graph
P [Gates] P [BENCH] P [Graph] P [Verify] P [Stats] P [Selection]
P' [BENCH] P' [Graph] P' [Verify] P' [Stats] P' [Replacements]

Example:
- Selected gates to replace
- Random static polygate
- No overhead limit

Key for the polygate circuit provided in the Keys TAB

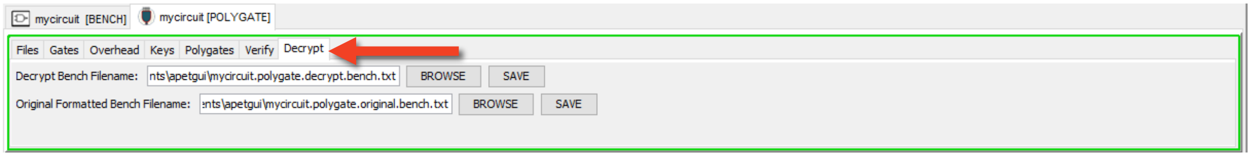

mycircuit [BENCH] | mycircuit [POLYGATE]

Files Gates Overhead Keys Polygates Verify Decrypt

Compress Key Key String Append Mode [✓ LSB] MSB

```
100011101110
```

Options to save a "decrypt" version of the polygate circuit that clearly shows key bit inputs - First click BROWSE for file path, then SAVE


mycircuit [BENCH] | mycircuit [POLYGATE]

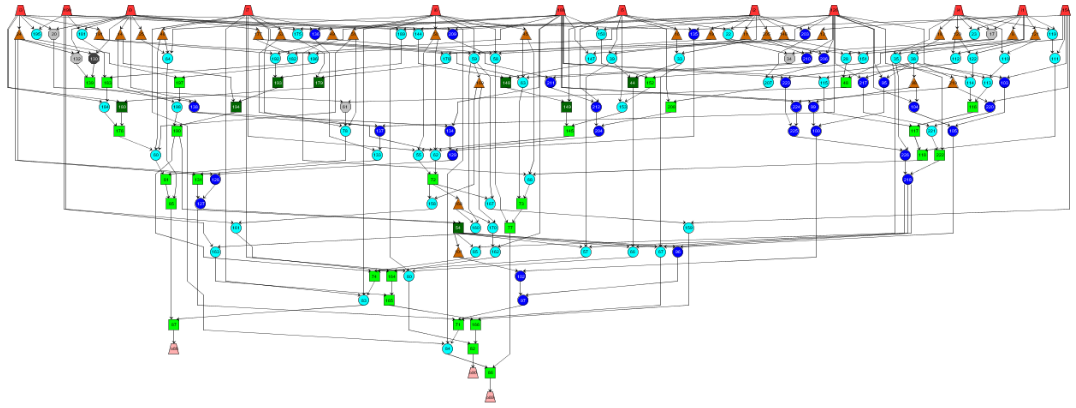
Files Gates Overhead Keys Polygates Verify Decrypt

Decrypt Bench Filename: nts\apetgui\mycircuit.polygate.decrypt.bench.txt BROWSE SAVE

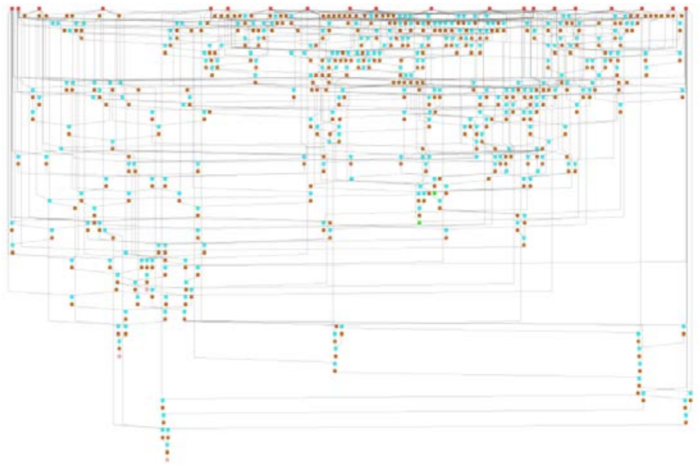
Original Formatted Bench Filename: :nts\apetgui\mycircuit.polygate.original.bench.txt BROWSE SAVE

Polygate Variants

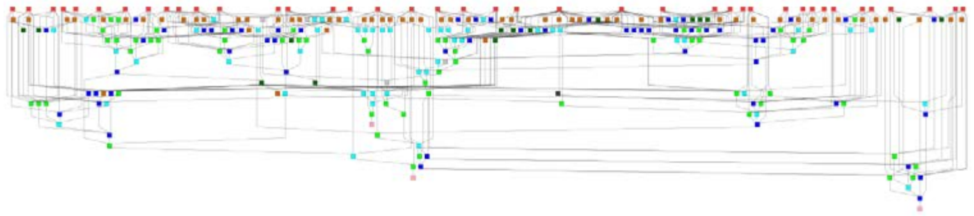
Key = 1010



Key = 101010100100

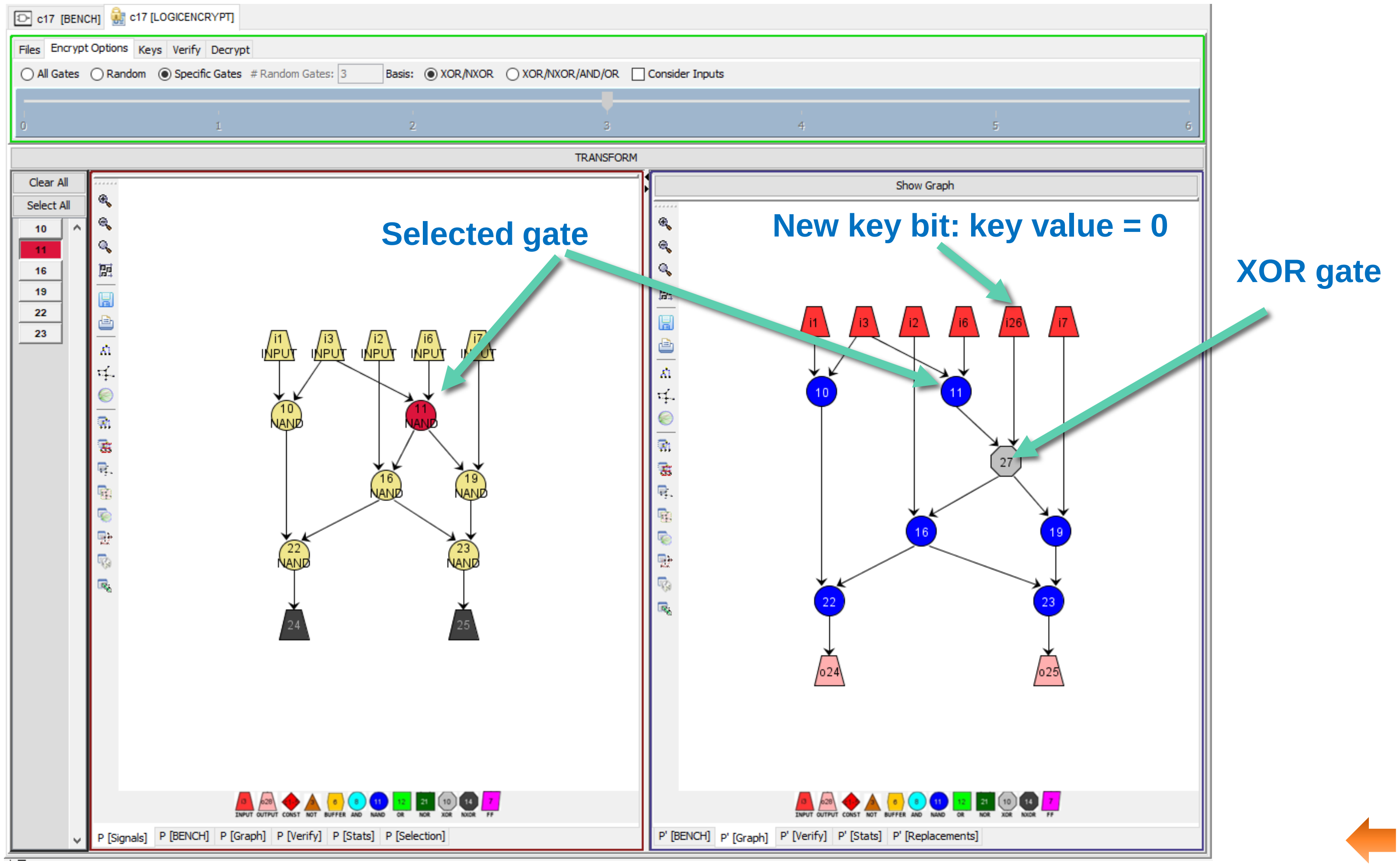


Key = 1011010111011101111001000011111



Logic Encryption

- Allows generation of variants using the standard/traditional algorithm for logic encryption based on insertion of XOR/NXOR gates and addition of a single key-bit input
- Also known as LOGIC LOCKING


c17 [BENCH] | c17 [LOGICENCRYPT]

Files | Encrypt Options | Keys | Verify | Decrypt

All Gates [✓ Specific Gates] Random · # Random Gates: 3 · Basis: [✓ XOR/NXOR] XOR/NXOR/AND/OR · Consider Inputs

TRANSFORM

Clear All · Select All

10, 11, 16, 19, 22, 23

Show Graph

P [Signals] · P [BENCH] · P [Graph] · P [Verify] · P [Stats] · P [Selection]

P' [BENCH] · P' [Graph] · P' [Verify] · P' [Stats] · P' [Replacements]

Logic Encryption

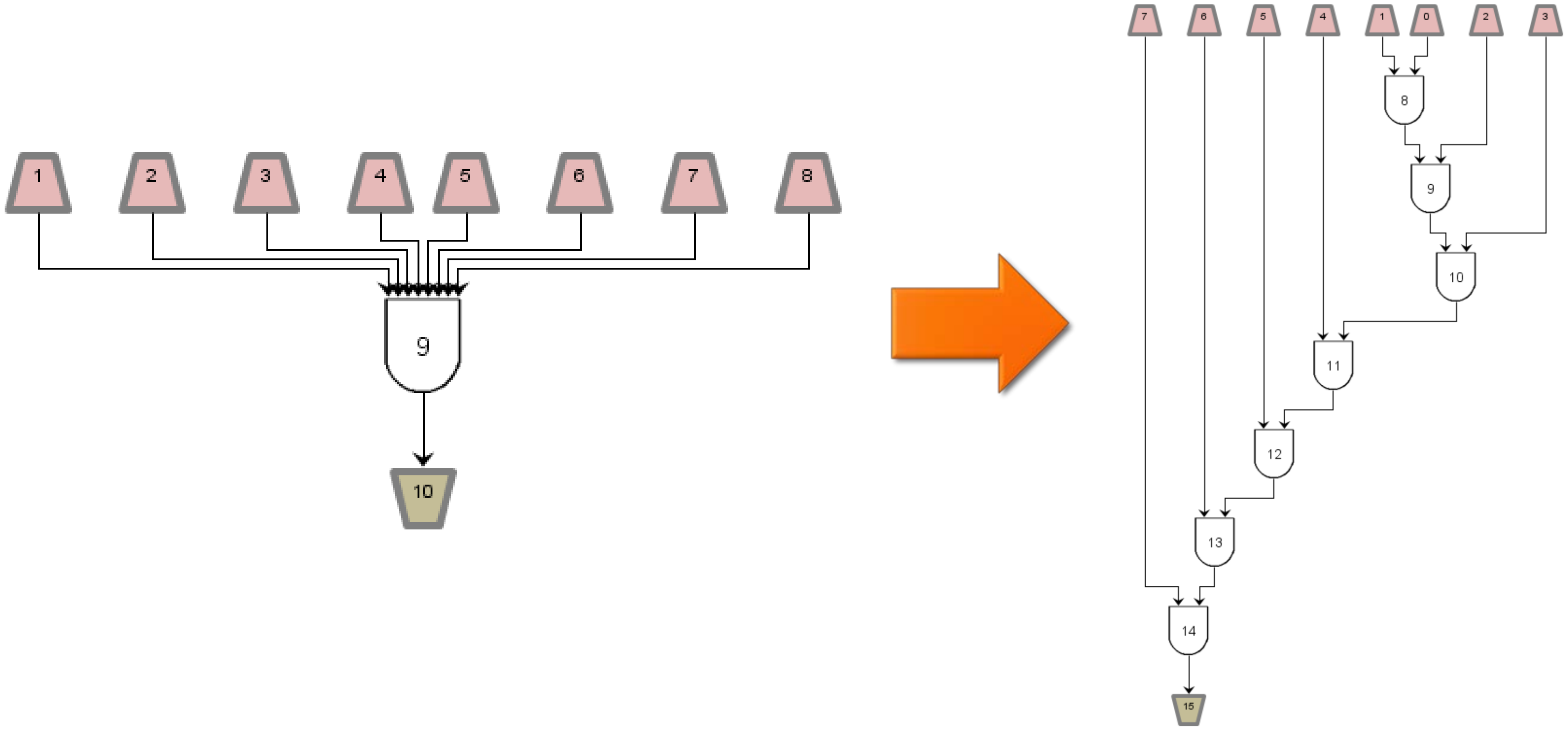
- Encrypt Options:
  - All gates
  - Some # of random gates
  - Specific gates

- Basis gate set types for insertion of logic locking :
  - XOR/NXOR
  - XOR/NXOR/AND/OR

- Key Options
  - Key compression
  - Append LSB/MSB mode

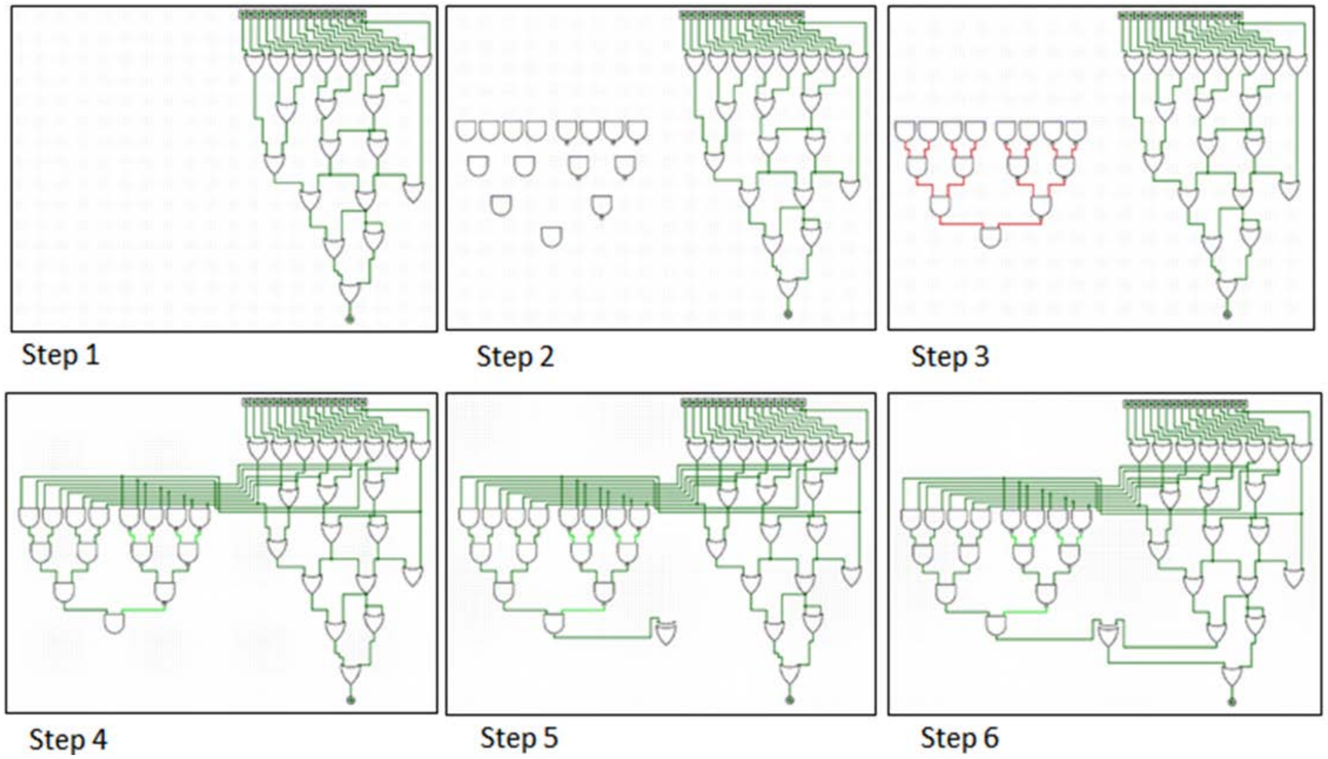
AND-Tree Insertion

- AND trees are essentially multiple-input AND gates that are typically decomposed:



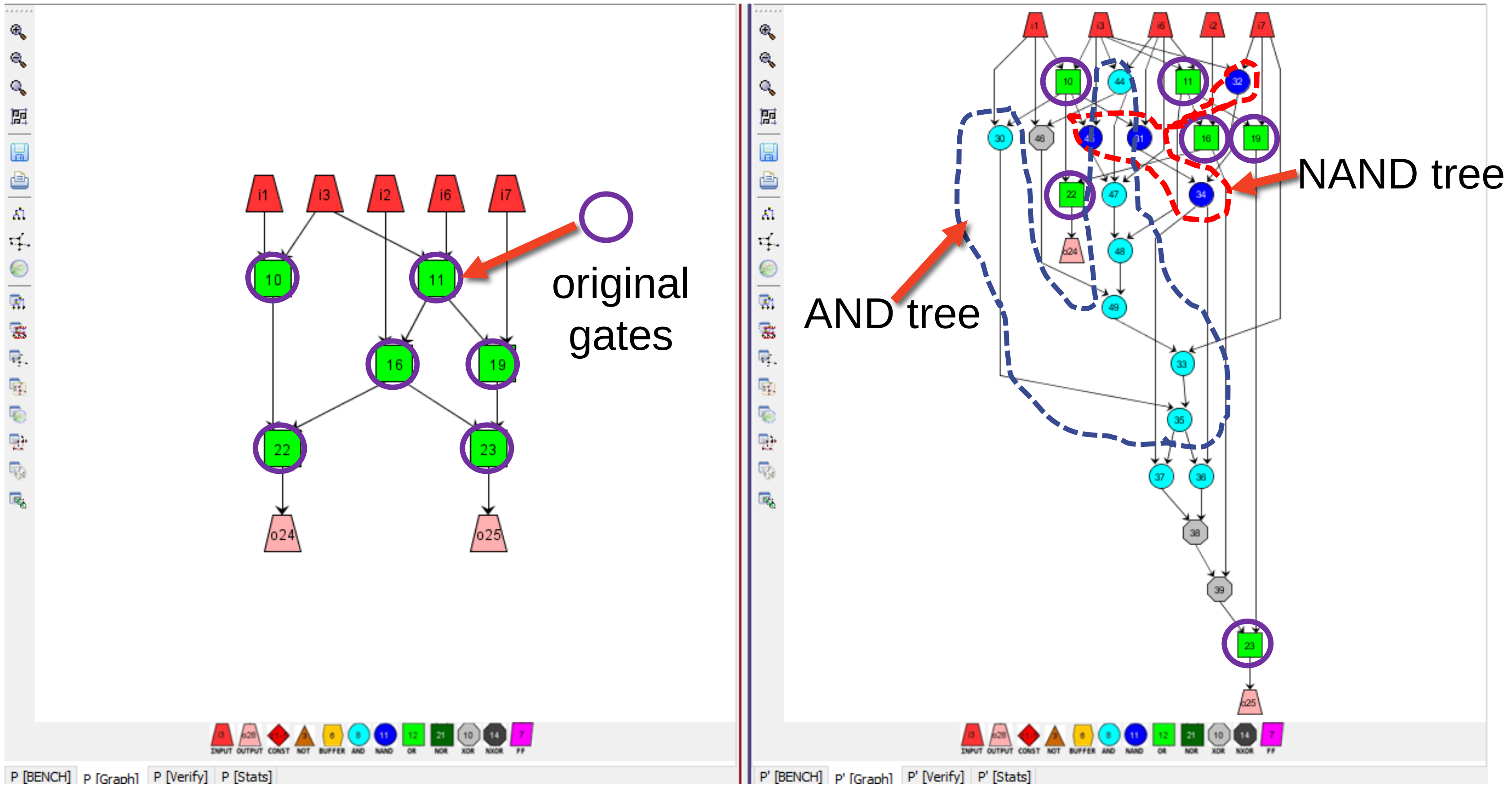
AND-Tree Insertion

- Inserting AND-tree structures are intended as a countermeasure to SAT-based reasoners by inserting a subcircuit that requires 2^{n-1} average evaluation
- The structure can be composed of alternating AND/AND or AND/NAND logic in parallel and then inserted (randomly) into a parent circuit in a semantically preserving manner
- Insertion approach is similar to logic locking where a predetermined 0/1 value produces semantically equivalent functions

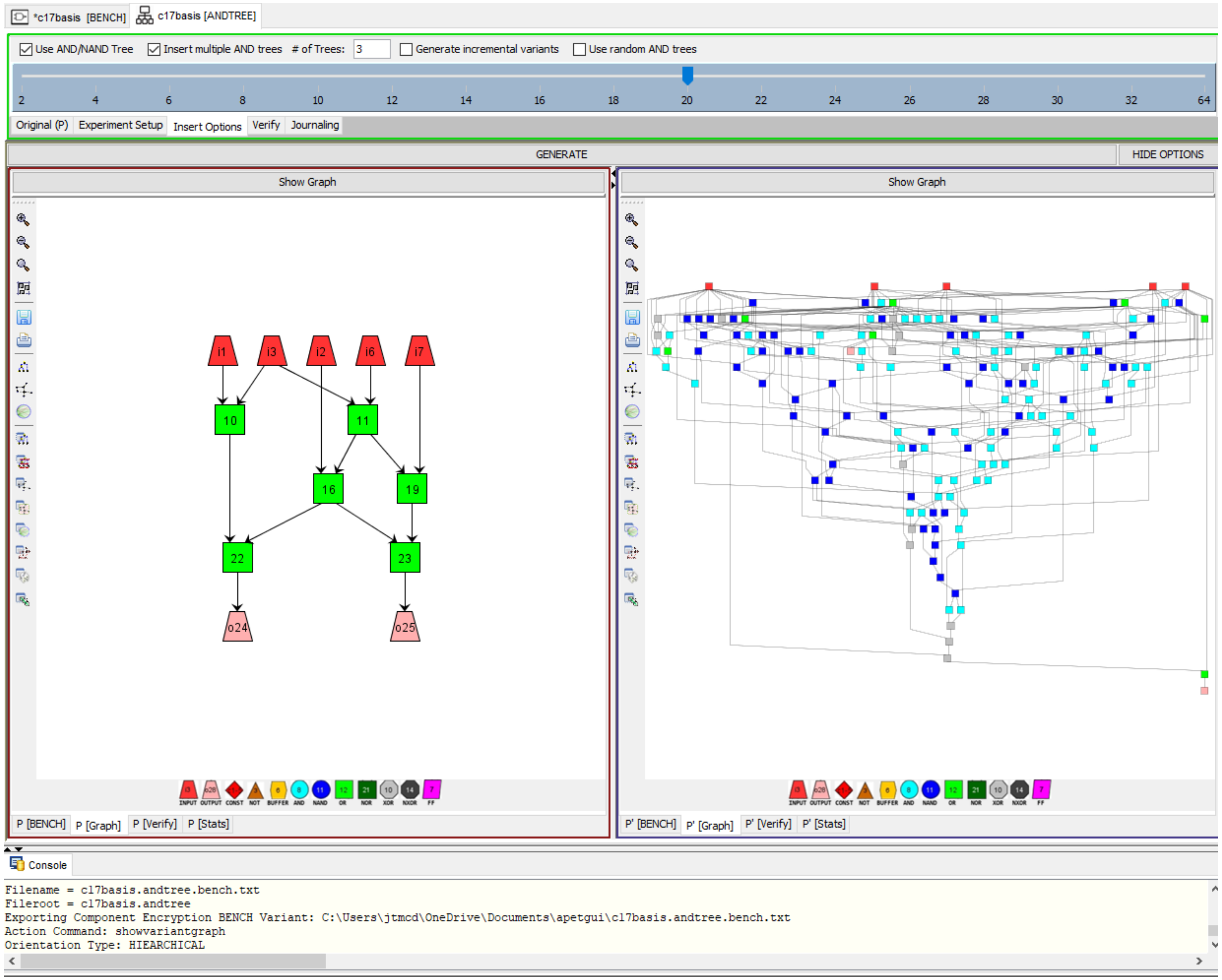


AND-Tree Insertion

- Paired AND/NAND trees can be inserted into a circuit and used for semantically equivalent insertion using XOR/NXOR logic and constant 0/1 signals



AND-Tree Experiments


*c17basis [BENCH]
c17basis [ANDTREE]
GENERATE
HIDE OPTIONS
Show Graph
i1
i3
i2
i6
i7
10
11
16
19
22
23
o24
o25
INPUT OUTPUT CONST NOT BUFFER AND NAND OR NOR XOR NXOR FF
P [BENCH] P [Graph] P [Verify] P [Stats]
P' [BENCH] P' [Graph] P' [Verify] P' [Stats]
Console

☑ Use AND/NAND Tree ☑ Insert multiple AND trees # of Trees: 3 ☐ Generate incremental variants ☐ Use random AND trees

2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 64

Original (P) | Experiment Setup | Insert Options | Verify | Journaling

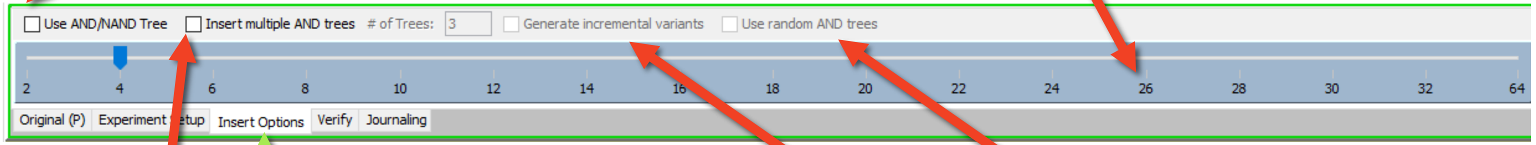
```
Filename = c17basis.andtree.bench.txt
Fileroot = c17basis.andtree
Exporting Component Encryption BENCH Variant: C:\Users\jtmcd\OneDrive\Documents\apetgui\c17basis.andtree.bench.txt
Action Command: showvariantgraph
Orientation Type: HIERARCHICAL
```

1) Experiment Setup: Set file and directory information

2) Insert Options:

Insert a Single AND Tree

- Use AND/NAND trees (vs. just balanced AND trees)
- Slider: input size of AND trees (2 to 64 input)



Insert multiple AND trees (Y/N)

- # of TREES
- Generate incremental variants (save variant after each AND tree is inserted, up to the final one)
- Use Random AND Trees: instead of a fixed input size AND tree, use random input size (between 2 to 64)

Reductions:

Equational Reducer

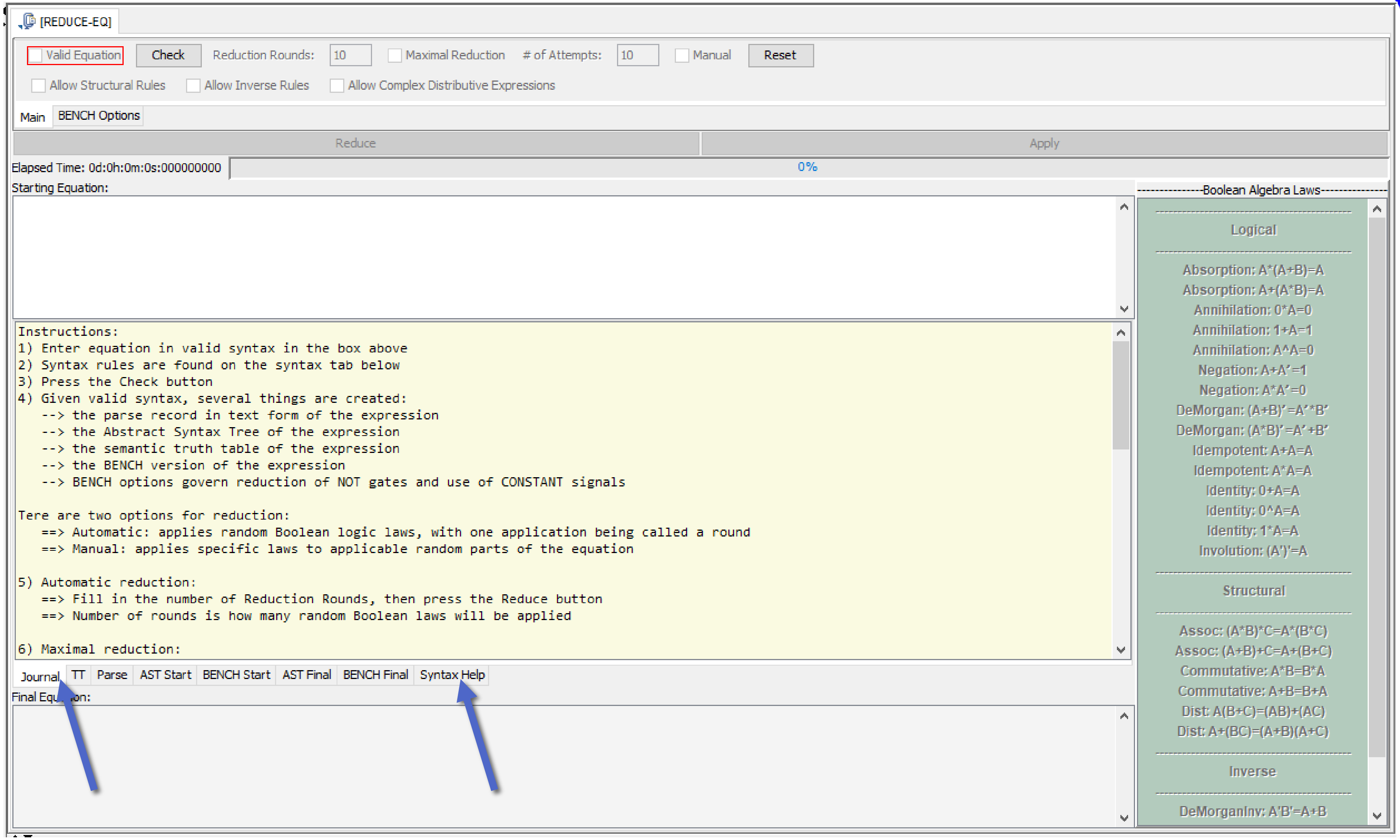
Pattern Based Circuit Reducer

Structural

Shaped

Pattern Viewer

Equational Reducer



```
Instructions:
1) Enter equation in valid syntax in the box above
2) Syntax rules are found on the syntax tab below
3) Press the Check button
4) Given valid syntax, several things are created:
   --> the parse record in text form of the expression
   --> the Abstract Syntax Tree of the expression
   --> the semantic truth table of the expression
   --> the BENCH version of the expression
   --> BENCH options govern reduction of NOT gates and use of CONSTANT signals

Tere are two options for reduction:
   ==> Automatic: applies random Boolean logic laws, with one application being called a round
   ==> Manual: applies specific laws to applicable random parts of the equation

5) Automatic reduction:
   ==> Fill in the number of Reduction Rounds, then press the Reduce button
   ==> Number of rounds is how many random Boolean laws will be applied

6) Maximal reduction:
```

-------------Boolean Algebra Laws-------------

Logical

- Absorption: A*(A+B)=A
- Absorption: A+(A*B)=A
- Annihilation: 0*A=0
- Annihilation: 1+A=1
- Annihilation: A^A=0
- Negation: A+A'=1
- Negation: A*A'=0
- DeMorgan: (A+B)'=A'*B'
- DeMorgan: (A*B)'=A'+B'
- Idempotent: A+A=A
- Idempotent: A*A=A
- Identity: 0+A=A
- Identity: 0^A=A
- Identity: 1*A=A
- Involution: (A')'=A

Structural

- Assoc: (A*B)*C=A*(B*C)
- Assoc: (A+B)+C=A+(B+C)
- Commutative: A*B=B*A
- Commutative: A+B=B+A
- Dist: A(B+C)=(AB)+(AC)
- Dist: A+(BC)=(A+B)(A+C)

Inverse

- DeMorganInv: A'B'=A+B

Equational Reducer

Instructions:
1) Enter equation in valid syntax in the box above

2) Syntax rules are found on the syntax tab below

3) Press the Check button

4) Given valid syntax, several things are created:
  --> the parse record in text form of the expression
  --> the Abstract Syntax Tree of the expression
  --> the semantic truth table of the expression
  --> the BENCH version of the expression
  --> BENCH options govern reduction of NOT gates and use of CONSTANT signals

There are two options for reduction:
  ==> Automatic: applies random Boolean logic laws, with one application being called a round
  ==> Manual: applies specific laws to applicable random parts of the equation

5) Automatic reduction:
  ==> Fill in the number of Reduction Rounds, then press the Reduce button
  ==> Number of rounds is how many random Boolean laws will be applied

6) Maximal reduction:
  ==> To run multiple attempts at reduction and save the optimal, check Maximal Reduction
  ==> Fill in the # of attempts, then press the Reduce button

7) Manual reduction:
  ==> To apply manual reductions, click the Manual checkbox
  ==> The possible reductions for the current expression are seen in the list on the right
  ==> Click on the reduction type and click the Apply button
  ==> Each reduction will present new options for reduction

Moving between manual and automatic will clear the journal and start at the original expression
Typing or modifying the Boolean equation will clear any AST, truth table, or BENCH and need to be rechecked
Click the Reset button to clear all panels

Equational Reducer

Three options can guide application of reduction rules: logical reductions are applied before commutativity, distributivity, associativity, and inverse laws

- Allow Structural will include commutative, associative, simple distributive, and inverse patterns that are possible
- Allow Inverse will include inverse patterns that are possible if no other structural ones are possible
- Allow Distributive will allow complex distributive patterns

Structural patterns include:

ReduceAssociativityType.VAR1_AND_VAR2_AND_VAR3
ReduceAssociativityType.VAR1_OR_VAR2_OR_VAR3
ReduceCommutativityType.VAR1_AND_VAR2
ReduceCommutativityType.VAR1_OR_VAR2

Distributive structural patterns include:

ReduceDistributivityType.VAR1_AND_VAR2_OR_VAR3
ReduceDistributivityType.VAR1_OR_VAR2_AND_VAR3

Inverse patterns include:

ReduceDeMorganInverseType.NOT_VAR1_AND_NOT_VAR2
ReduceDeMorganInverseType.NOT_VAR1_OR_NOT_VAR2
ReduceDistributivityInverseType.VAR1_AND_VAR2_OR_VAR1_AND_VAR3
ReduceDistributivityInverseType.VAR1_OR_VAR2_AND_VAR1_OR_VAR3

Equational Reducer Syntax

Equations should take the form of:

OUTVAR = EQUATION

- OUTVAR must be of the form: oX => o0, o1, o2, etc.
- EQUATION is a combination of VARIABLES and OPERATORS.
- VARIABLES must be of the form iX => i0, i1, i2, etc.
- VARIABLES are ordered in circuit input by number
- OPERATORS must be one of: '(NOT) +(OR) *(AND) ^(XOR)
- Constant Zeros (0) / Ones(1) are allowed as VARIABLES

General rules:

- At least 1 VARIABLE required (o1 = 0/o1 = 1 not allowed)
- Use parenthesis to clarify logical expressions and precidence

Examples:

```
o1 = i0 + i1
o1 = ((i0' * i1)' + (i2 * i3')')'
o1 = i1 * 1; o2 = i4 + i18
o0 = i1 + i2 ^ i3 * i4
o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))'
o6 = (i7 * i9) + i1
o12 = (i25 ^ i512)'
```

Precedence rules:

- Parenthesis have highest precedence
- NOT (') associates to the left before other OPERATORS
- AND (*) associates before OR (+) and XOR (^)
- XOR (^) associates before OR (+)

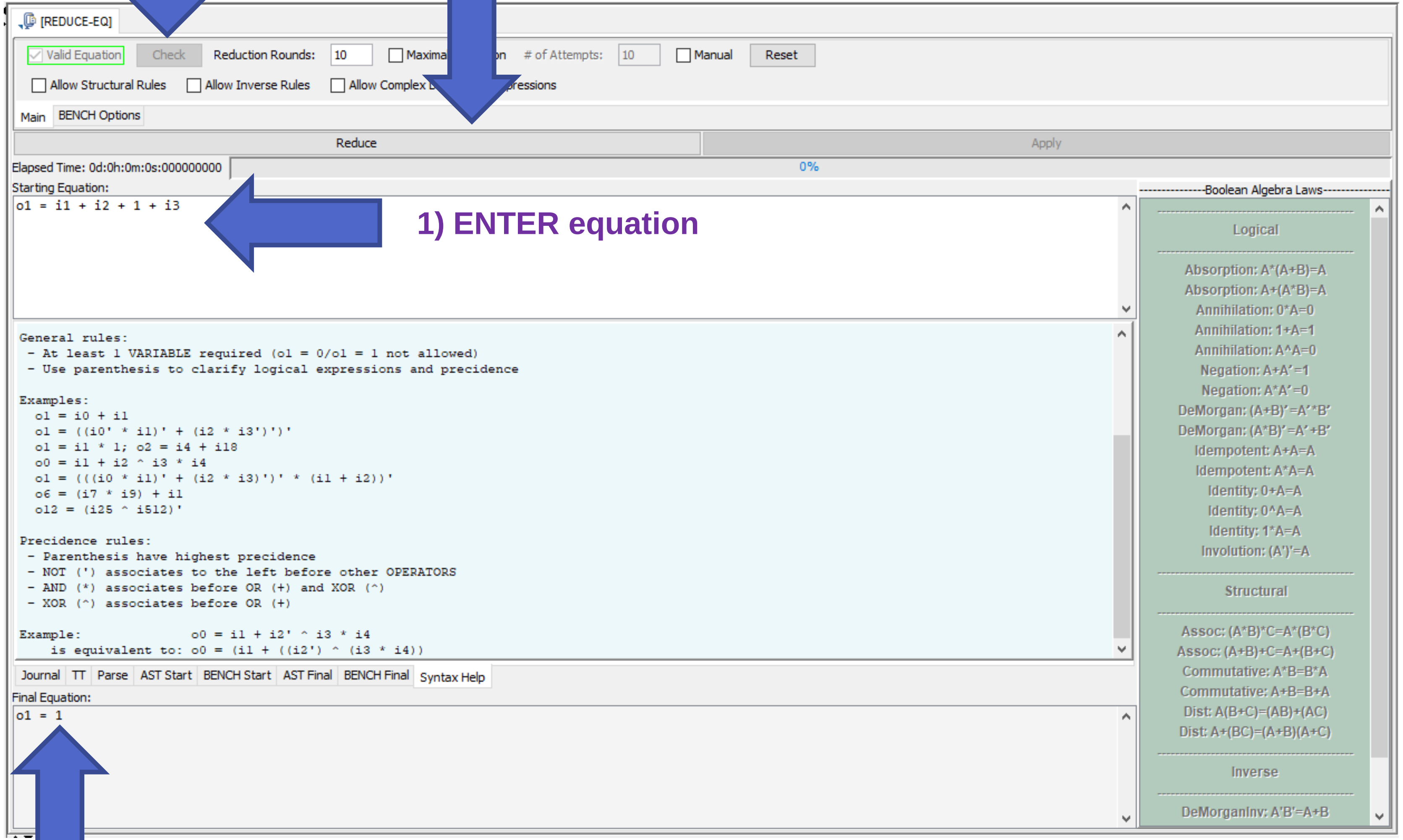
Example: o0 = i1 + i2' ^ i3 * i4

is equivalent to: o0 = (i1 + ((i2') ^ (i3 * i4))

Reducer Experiments

2) Check

3) Click Reduce



Elapsed Time: 0d:0h:0m:0s:000000000

Starting Equation:

```
o1 = i1 + i2 + 1 + i3
```

1) ENTER equation

```
General rules:
 - At least 1 VARIABLE required (o1 = 0/o1 = 1 not allowed)
 - Use parenthesis to clarify logical expressions and precidence

Examples:
  o1 = i0 + i1
  o1 = ((i0' * i1)' + (i2 * i3')')'
  o1 = i1 * 1; o2 = i4 + i18
  o0 = i1 + i2 ^ i3 * i4
  o1 = (((i0 * i1)' + (i2 * i3)')' * (i1 + i2))'
  o6 = (i7 * i9) + i1
  o12 = (i25 ^ i512)'

Precidence rules:
 - Parenthesis have highest precidence
 - NOT (') associates to the left before other OPERATORS
 - AND (*) associates before OR (+) and XOR (^)
 - XOR (^) associates before OR (+)

Example:              o0 = i1 + i2' ^ i3 * i4
    is equivalent to: o0 = (i1 + ((i2') ^ (i3 * i4))
```

Final Equation:

```
o1 = 1
```

Boolean Algebra Laws

Logical

- Absorption: A*(A+B)=A
- Absorption: A+(A*B)=A
- Annihilation: 0*A=0
- Annihilation: 1+A=1
- Annihilation: A^A=0
- Negation: A+A'=1
- Negation: A*A'=0
- DeMorgan: (A+B)'=A'*B'
- DeMorgan: (A*B)'=A'+B'
- Idempotent: A+A=A
- Idempotent: A*A=A
- Identity: 0+A=A
- Identity: 0^A=A
- Identity: 1*A=A
- Involution: (A')'=A

Structural

- Assoc: (A*B)*C=A*(B*C)
- Assoc: (A+B)+C=A+(B+C)
- Commutative: A*B=B*A
- Commutative: A+B=B+A
- Dist: A(B+C)=(AB)+(AC)
- Dist: A+(BC)=(A+B)(A+C)

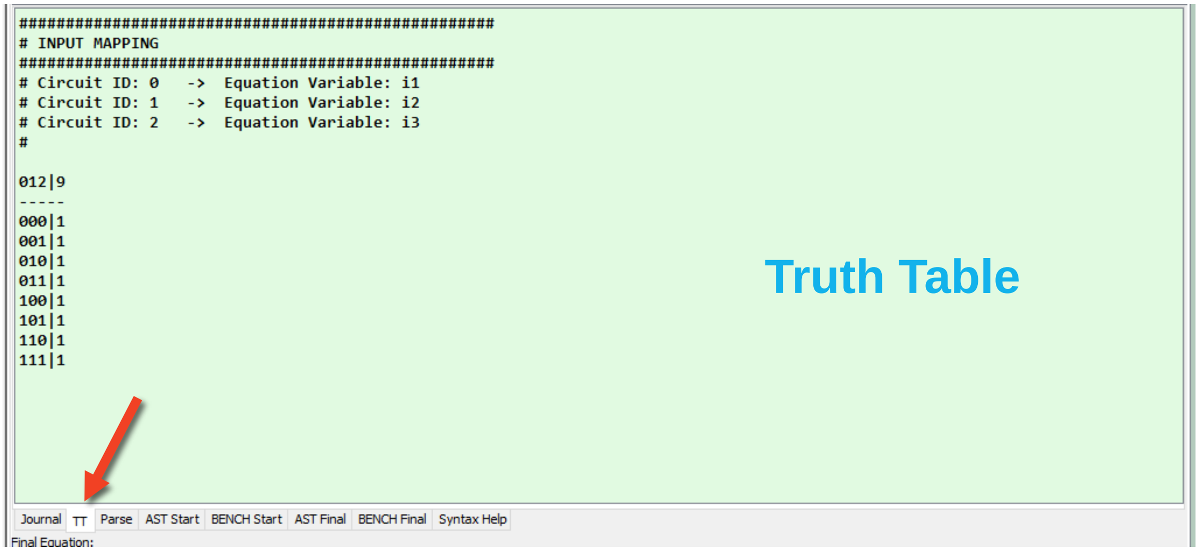
Inverse

- DeMorganInv: A'B'=A+B

4) Final Equation

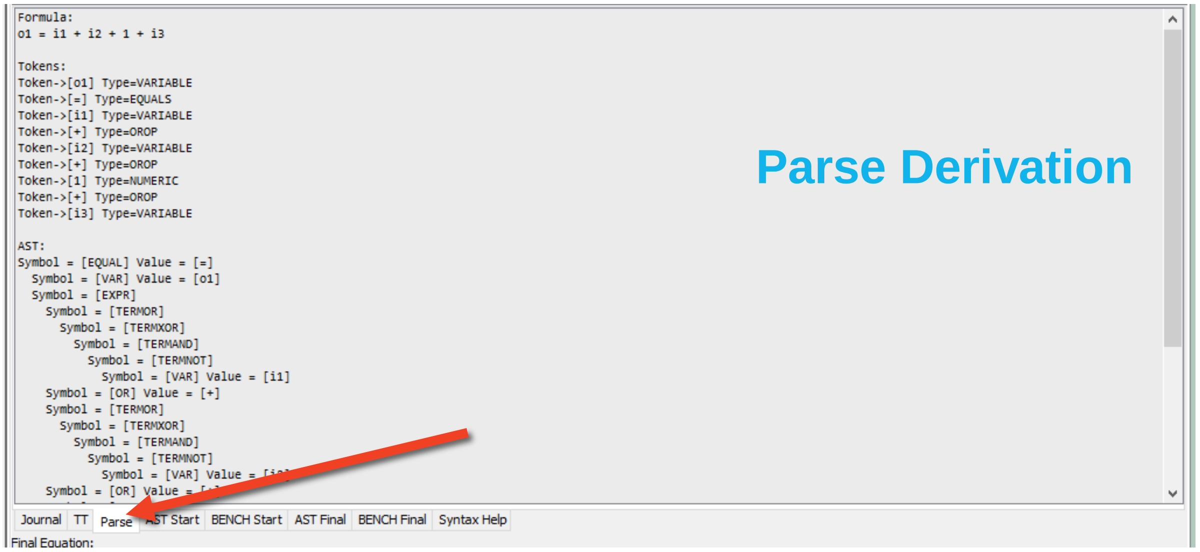
Reducer Experiments

Pre Reduction Views:



```
####################################################
# INPUT MAPPING
####################################################
# Circuit ID: 0   ->  Equation Variable: i1
# Circuit ID: 1   ->  Equation Variable: i2
# Circuit ID: 2   ->  Equation Variable: i3
#

012|9
-----
000|1
001|1
010|1
011|1
100|1
101|1
110|1
111|1
```

Final Equation:

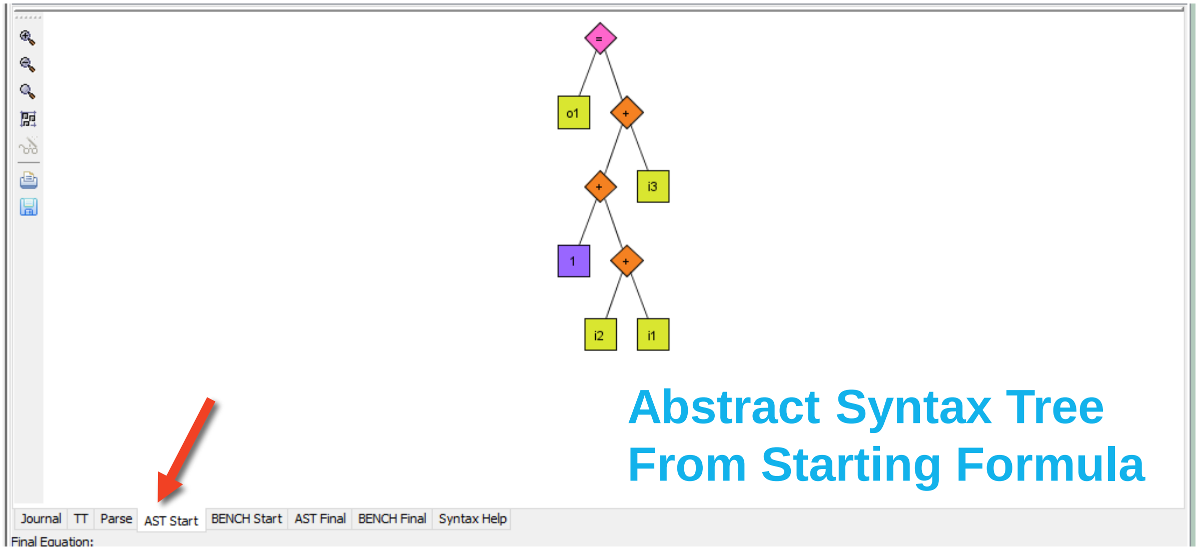


```
Formula:
o1 = i1 + i2 + 1 + i3

Tokens:
Token->[o1] Type=VARIABLE
Token->[=] Type=EQUALS
Token->[i1] Type=VARIABLE
Token->[+] Type=OROP
Token->[i2] Type=VARIABLE
Token->[+] Type=OROP
Token->[1] Type=NUMERIC
Token->[+] Type=OROP
Token->[i3] Type=VARIABLE

AST:
Symbol = [EQUAL] Value = [=]
  Symbol = [VAR] Value = [o1]
  Symbol = [EXPR]
    Symbol = [TERMOR]
      Symbol = [TERMXOR]
        Symbol = [TERMAND]
          Symbol = [TERMNOT]
            Symbol = [VAR] Value = [i1]
    Symbol = [OR] Value = [+]
    Symbol = [TERMOR]
      Symbol = [TERMXOR]
        Symbol = [TERMAND]
          Symbol = [TERMNOT]
            Symbol = [VAR] Value = [i
    Symbol = [OR] Value = [
```

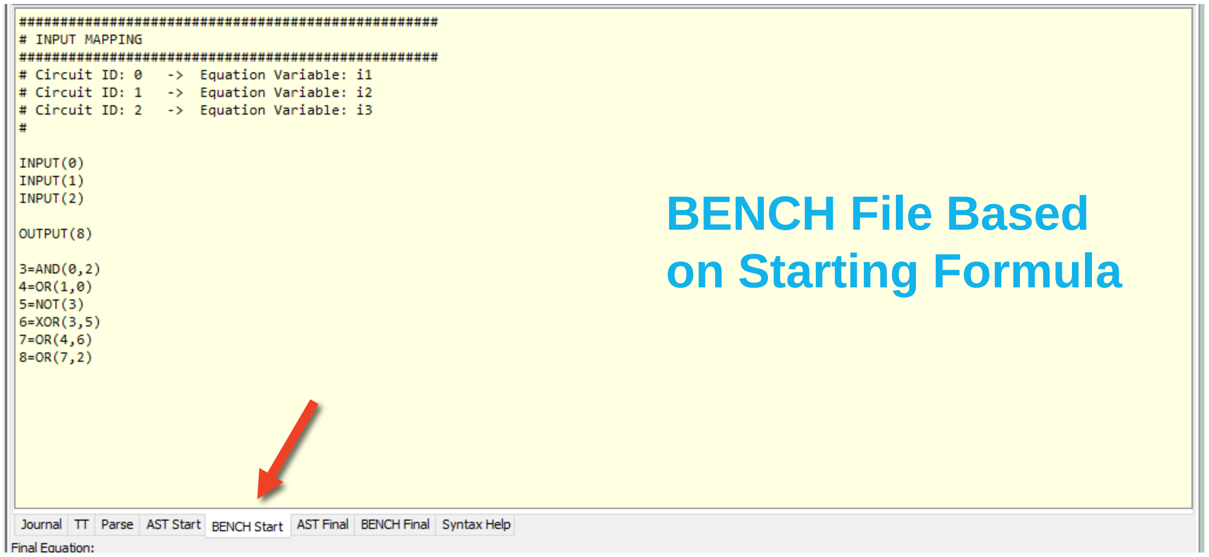
Final Equation:

Reducer Experiments

Pre Reduction Views:



Abstract Syntax Tree From Starting Formula

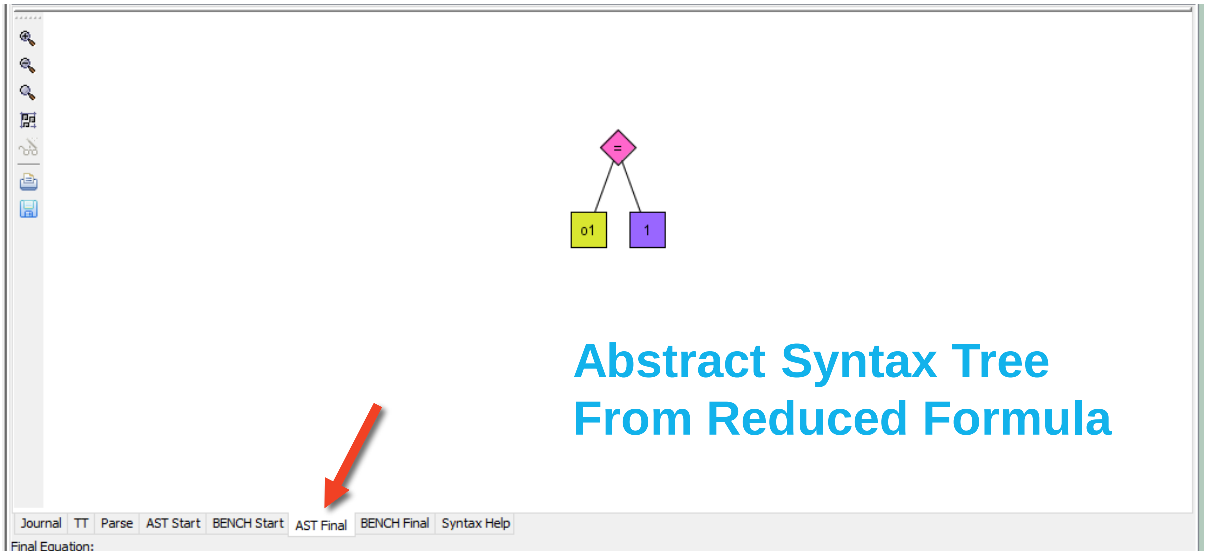


```
#####################################################
# INPUT MAPPING
#####################################################
# Circuit ID: 0    ->  Equation Variable: i1
# Circuit ID: 1    ->  Equation Variable: i2
# Circuit ID: 2    ->  Equation Variable: i3
#

INPUT(0)
INPUT(1)
INPUT(2)

OUTPUT(8)

3=AND(0,2)
4=OR(1,0)
5=NOT(3)
6=XOR(3,5)
7=OR(4,6)
8=OR(7,2)
```

BENCH File Based on Starting Formula

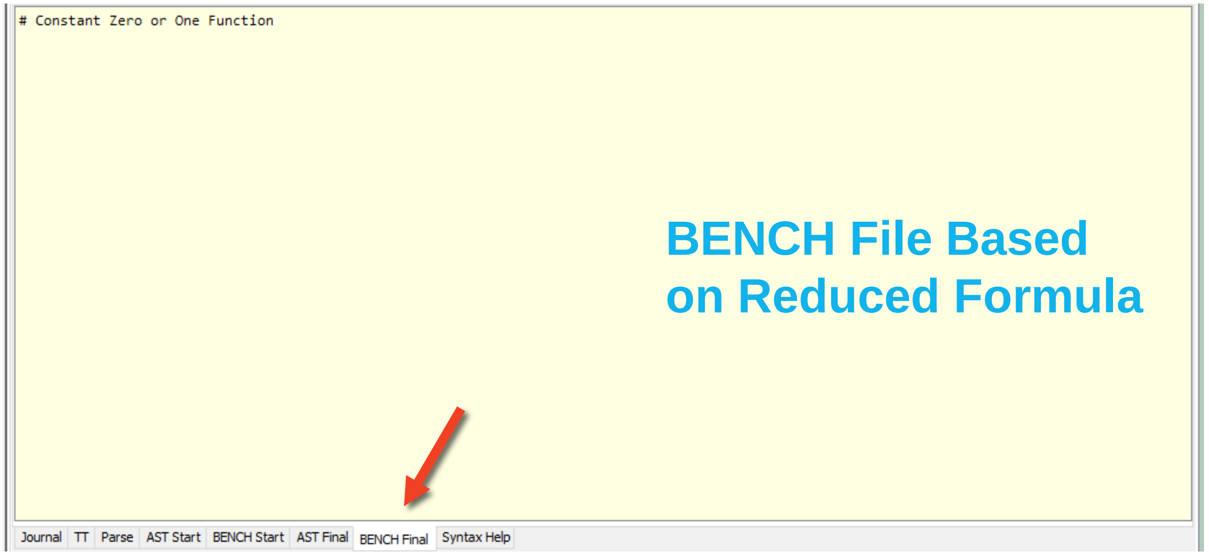
Reducer Experiments

Post Reduction Views:



Final Equation:

Abstract Syntax Tree From Reduced Formula

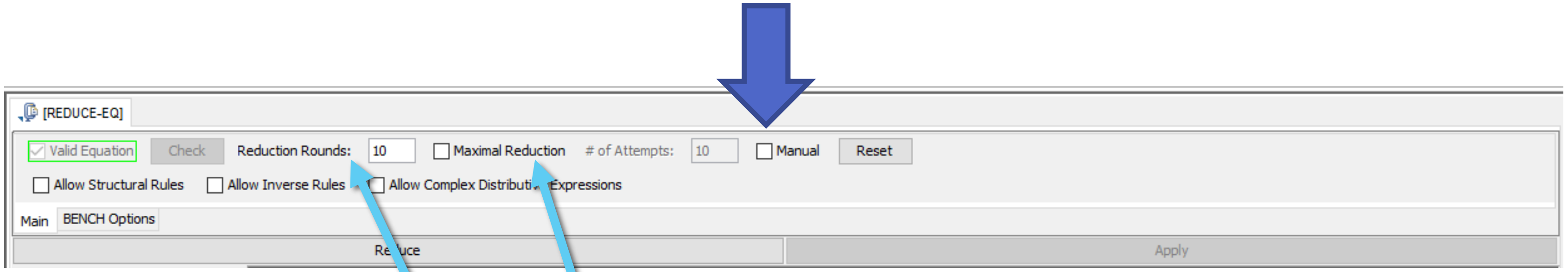


```
# Constant Zero or One Function
```

BENCH File Based on Reduced Formula

Reducer Experiments

Main Option: Automatic vs. Manual


[REDUCE-EQ]

Valid Equation | Check | Reduction Rounds: 10 | Maximal Reduction | # of Attempts: 10 | Manual | Reset

Allow Structural Rules | Allow Inverse Rules | Allow Complex Distributive Expressions

Main | BENCH Options

Reduce | Apply

Automated

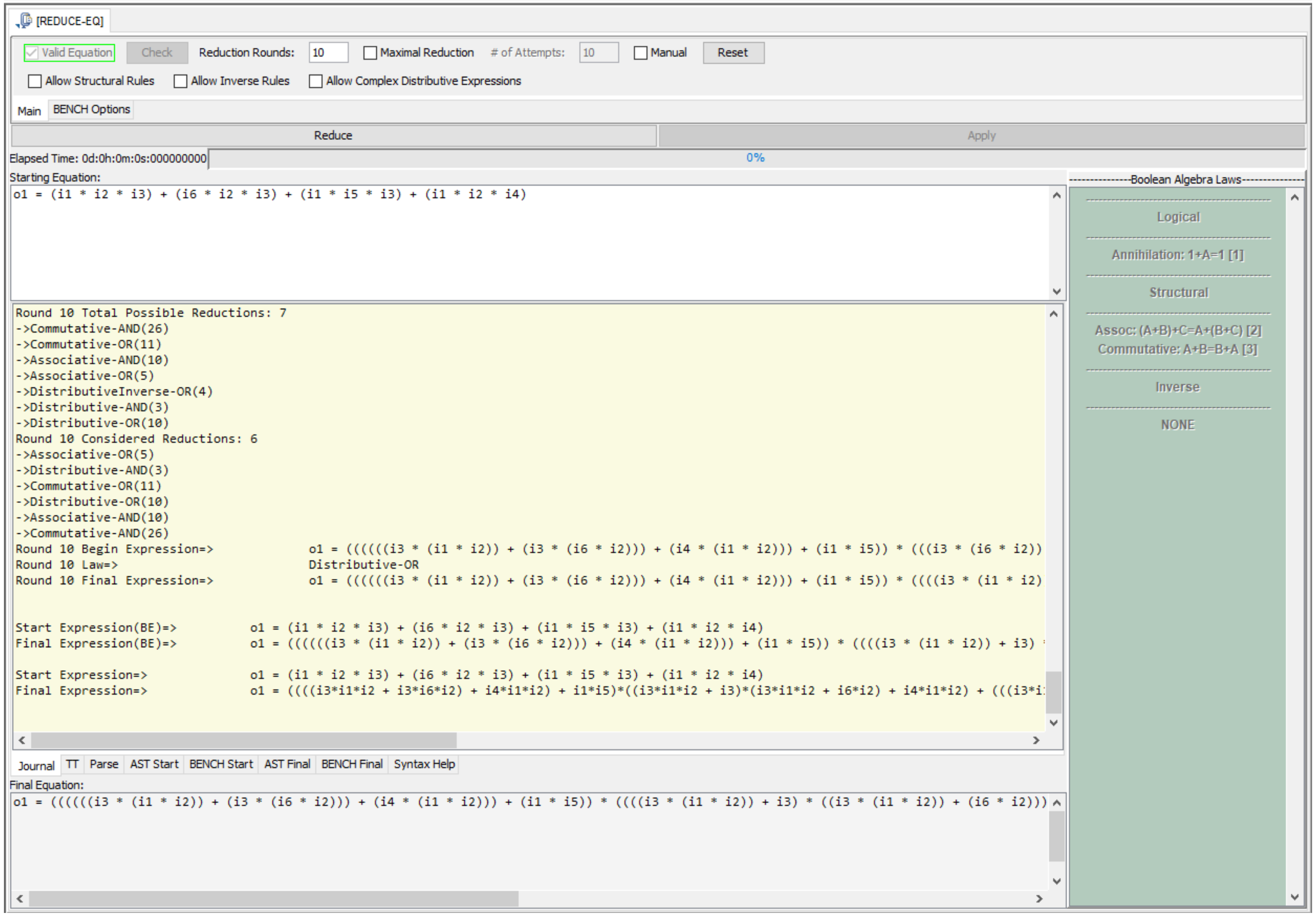
- # of Reduction Rounds
- Maximal Reduction (Y/N): unlimited rounds until the expression cannot be reduced further

Automated reduction is non-deterministic: Boolean logic laws are applied randomly and thus different results may be obtained depending on the order and specific sequence

There may be multiple statements which can be reduced by the same appropriate logic law: these are also chosen randomly

Reducer Experiments

- Example: 10 Reduction Rounds



Reduction Rounds: 10 ☐ Maximal Reduction # of Attempts: 10 ☐ Manual

☐ Allow Structural Rules ☐ Allow Inverse Rules ☐ Allow Complex Distributive Expressions

Elapsed Time: 0d:0h:0m:0s:000000000

Starting Equation:

```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

```
Round 10 Total Possible Reductions: 7
->Commutative-AND(26)
->Commutative-OR(11)
->Associative-AND(10)
->Associative-OR(5)
->DistributiveInverse-OR(4)
->Distributive-AND(3)
->Distributive-OR(10)
Round 10 Considered Reductions: 6
->Associative-OR(5)
->Distributive-AND(3)
->Commutative-OR(11)
->Distributive-OR(10)
->Associative-AND(10)
->Commutative-AND(26)
Round 10 Begin Expression=>          o1 = ((((((i3 * (i1 * i2)) + (i3 * (i6 * i2))) + (i4 * (i1 * i2))) + (i1 * i5)) * (((i3 * (i6 * i2))
Round 10 Law=>                       Distributive-OR
Round 10 Final Expression=>          o1 = ((((((i3 * (i1 * i2)) + (i3 * (i6 * i2))) + (i4 * (i1 * i2))) + (i1 * i5)) * ((((i3 * (i1 * i2)


Start Expression(BE)=>          o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression(BE)=>          o1 = ((((((i3 * (i1 * i2)) + (i3 * (i6 * i2))) + (i4 * (i1 * i2))) + (i1 * i5)) * ((((i3 * (i1 * i2)) + i3)

Start Expression=>              o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression=>              o1 = ((((i3*i1*i2 + i3*i6*i2) + i4*i1*i2) + i1*i5)*((i3*i1*i2 + i3)*(i3*i1*i2 + i6*i2) + i4*i1*i2) + (((i3*i
```

Final Equation:

```
o1 = ((((((i3 * (i1 * i2)) + (i3 * (i6 * i2))) + (i4 * (i1 * i2))) + (i1 * i5)) * ((((i3 * (i1 * i2)) + i3) * ((i3 * (i1 * i2)) + (i6 * i2)))
```

-----Boilerplate Algebra Laws-----

Logical

Annihilation: 1+A=1 [1]

Structural

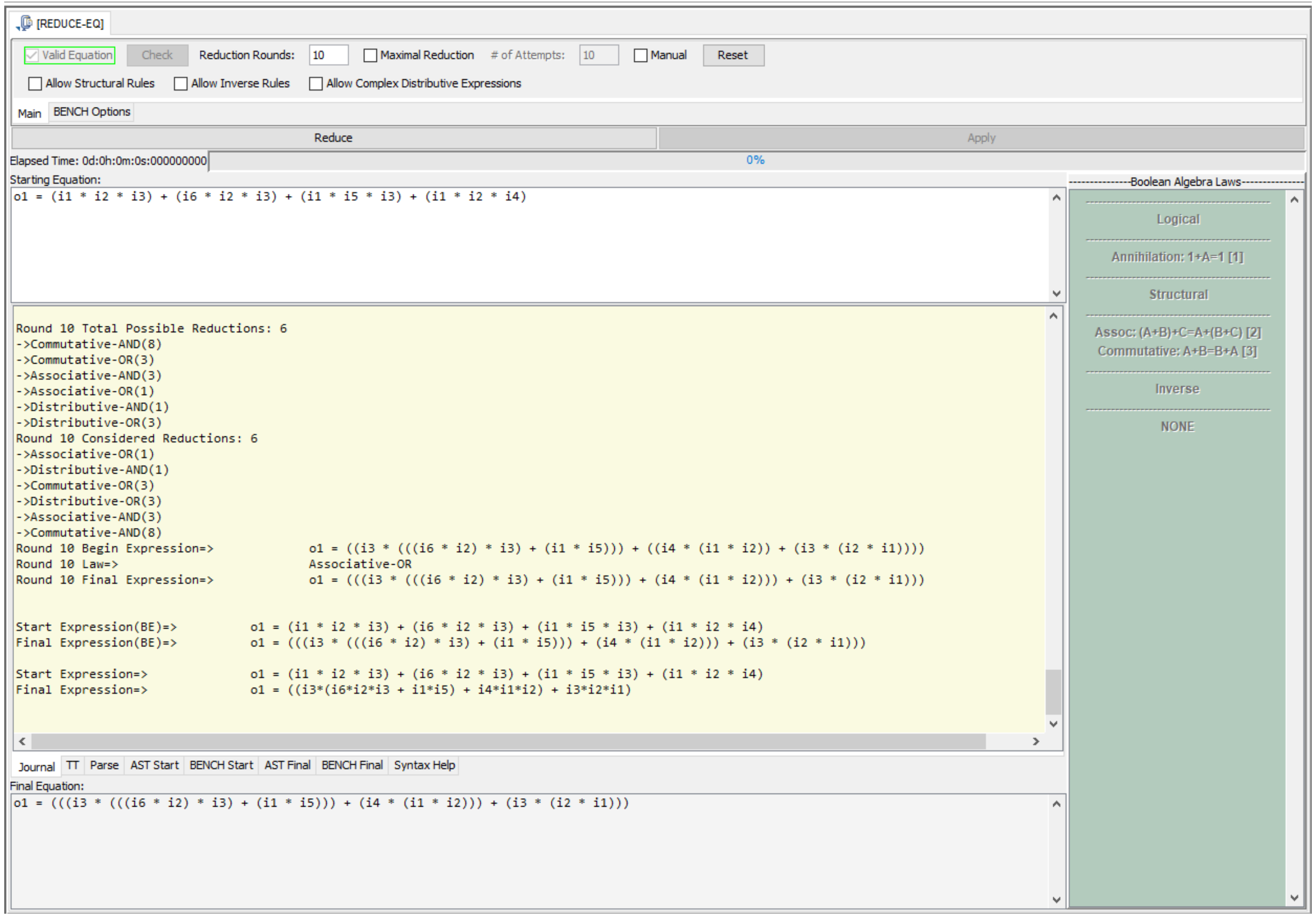
Assoc: (A+B)+C=A+(B+C) [2]
Commutative: A+B=B+A [3]

Inverse

NONE

Reducer Experiments

- Continuing to Click Reduce will produce a new result…
- Example: Same Equation, 10 Reduction Rounds, Different Results (smaller equation)



Elapsed Time: 0d:0h:0m:0s:000000000

Starting Equation:

```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

```
Round 10 Total Possible Reductions: 6
->Commutative-AND(8)
->Commutative-OR(3)
->Associative-AND(3)
->Associative-OR(1)
->Distributive-AND(1)
->Distributive-OR(3)
Round 10 Considered Reductions: 6
->Associative-OR(1)
->Distributive-AND(1)
->Commutative-OR(3)
->Distributive-OR(3)
->Associative-AND(3)
->Commutative-AND(8)
Round 10 Begin Expression=>           o1 = ((i3 * (((i6 * i2) * i3) + (i1 * i5))) + ((i4 * (i1 * i2)) + (i3 * (i2 * i1))))
Round 10 Law=>                        Associative-OR
Round 10 Final Expression=>           o1 = (((i3 * (((i6 * i2) * i3) + (i1 * i5))) + (i4 * (i1 * i2))) + (i3 * (i2 * i1)))


Start Expression(BE)=>        o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression(BE)=>        o1 = (((i3 * (((i6 * i2) * i3) + (i1 * i5))) + (i4 * (i1 * i2))) + (i3 * (i2 * i1)))

Start Expression=>            o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression=>            o1 = ((i3*(i6*i2*i3 + i1*i5) + i4*i1*i2) + i3*i2*i1)
```

Final Equation:

```
o1 = (((i3 * (((i6 * i2) * i3) + (i1 * i5))) + (i4 * (i1 * i2))) + (i3 * (i2 * i1)))
```

---Boolean Algebra Laws---

Logical

Annihilation: 1+A=1 [1]

Structural

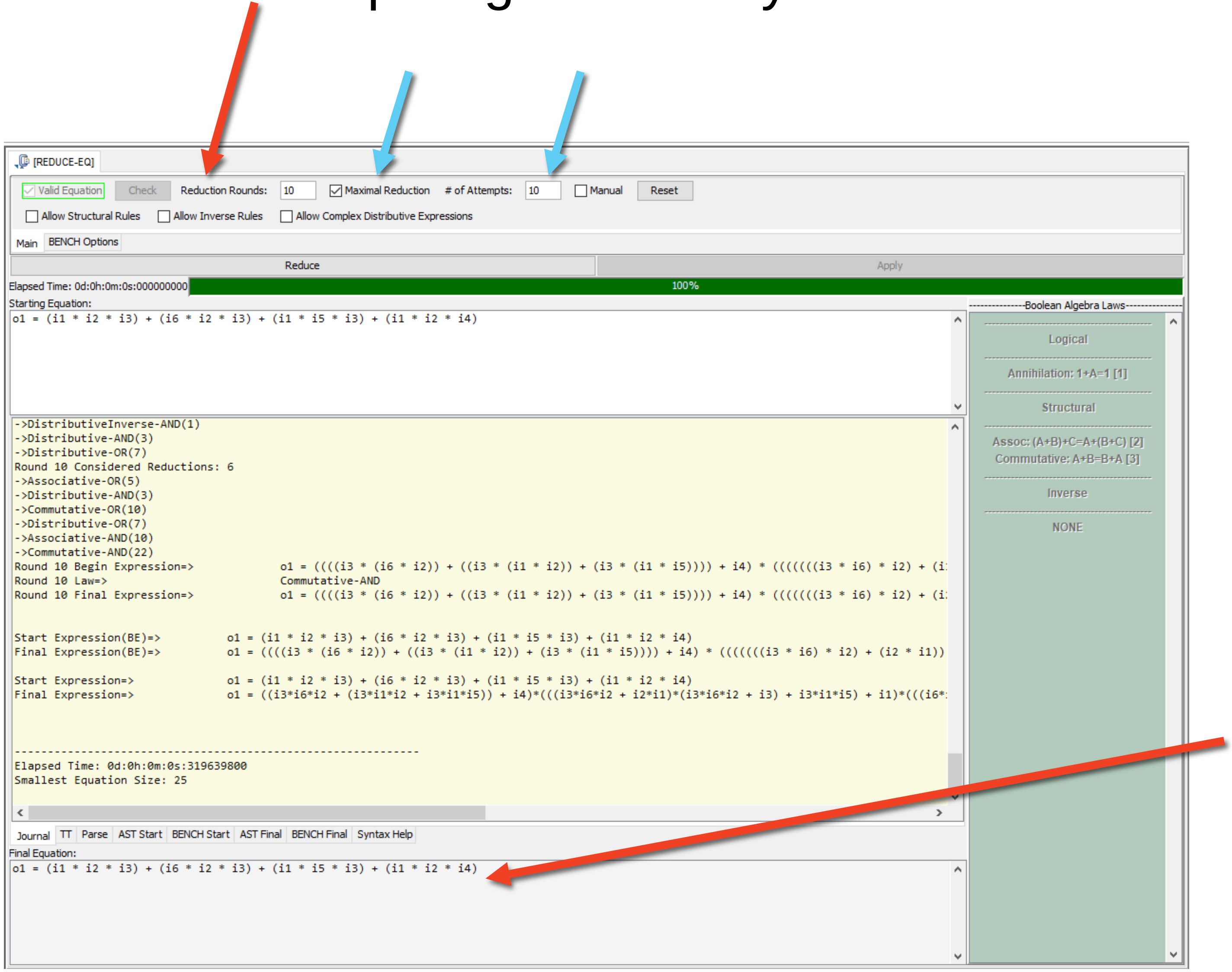
Assoc: (A+B)+C=A+(B+C) [2]
Commutative: A+B=B+A [3]

Inverse

NONE

Reducer Experiments

- Maximal Reduction: # of Attempts
- Each attempt is governed by # of Reduction Rounds



Settings: Reduction Rounds: 10; [✓ Maximal Reduction]; # of Attempts: 10

Elapsed Time: 0d:0h:0m:0s:000000000

Starting Equation:

```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

```
->DistributiveInverse-AND(1)
->Distributive-AND(3)
->Distributive-OR(7)
Round 10 Considered Reductions: 6
->Associative-OR(5)
->Distributive-AND(3)
->Commutative-OR(10)
->Distributive-OR(7)
->Associative-AND(10)
->Commutative-AND(22)
Round 10 Begin Expression=>          o1 = ((((i3 * (i6 * i2)) + ((i3 * (i1 * i2)) + (i3 * (i1 * i5)))) + i4) * (((((((i3 * i6) * i2) + (i
Round 10 Law=>                       Commutative-AND
Round 10 Final Expression=>          o1 = ((((i3 * (i6 * i2)) + ((i3 * (i1 * i2)) + (i3 * (i1 * i5)))) + i4) * (((((((i3 * i6) * i2) + (i


Start Expression(BE)=>        o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression(BE)=>        o1 = ((((i3 * (i6 * i2)) + ((i3 * (i1 * i2)) + (i3 * (i1 * i5)))) + i4) * (((((((i3 * i6) * i2) + (i2 * i1))

Start Expression=>            o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression=>            o1 = ((i3*i6*i2 + (i3*i1*i2 + i3*i1*i5)) + i4)*(((i3*i6*i2 + i2*i1)*(i3*i6*i2 + i3) + i3*i1*i5) + i1)*(((i6*



-----------------------------------------------------------------
Elapsed Time: 0d:0h:0m:0s:319639800
Smallest Equation Size: 25
```

Final Equation:

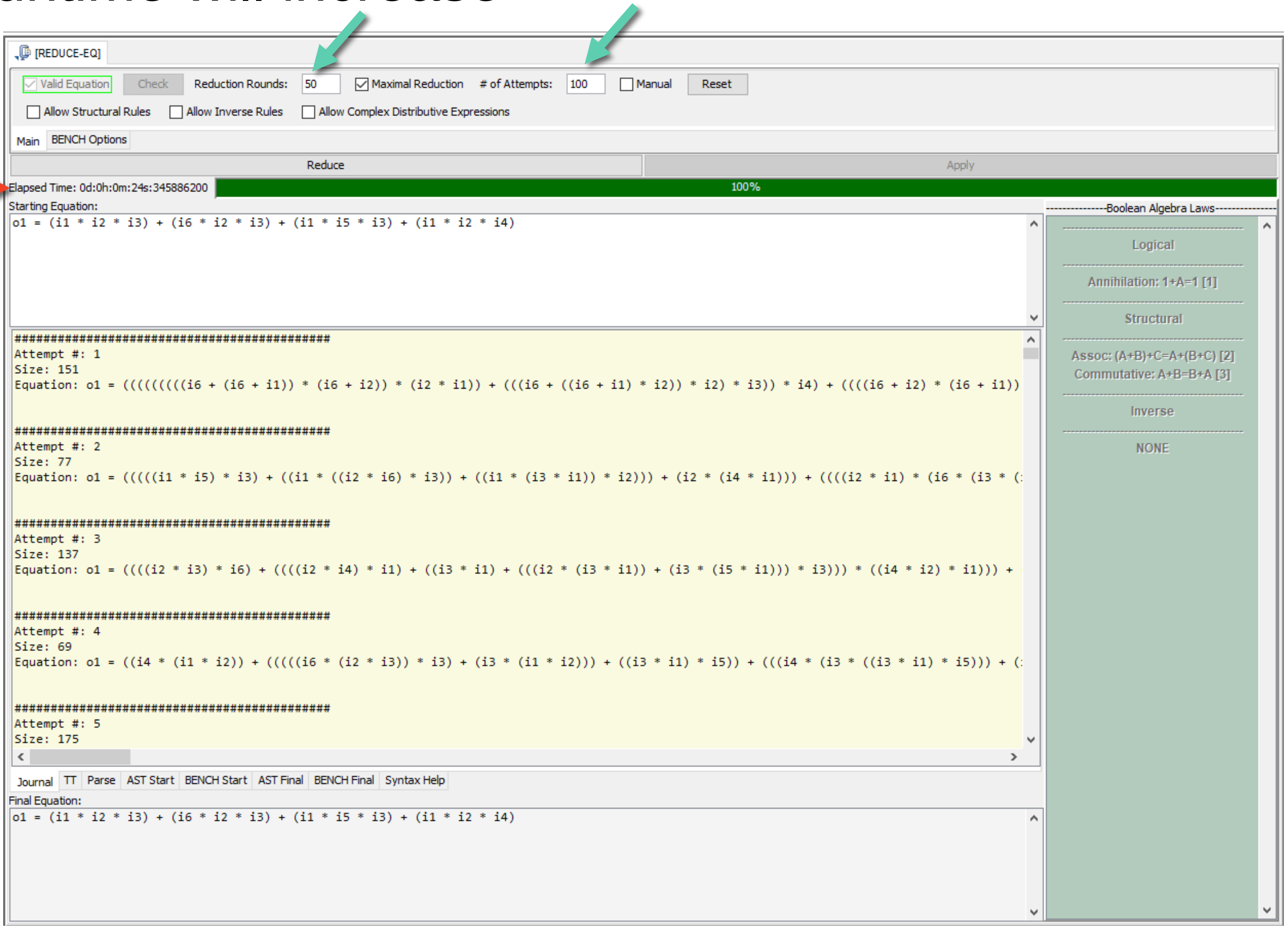
```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

Journal shows the best result of applying logic laws (smallest equation size)

If no attempt can make equation smaller, final equation is the original

Reducer Experiments

- Increasing # of rounds and # of attempts may (or may not) produce better results
- Runtime will increase



Reduction Rounds: 50 [✓ Maximal Reduction] # of Attempts: 100 [ ] Manual

[ ] Allow Structural Rules [ ] Allow Inverse Rules [ ] Allow Complex Distributive Expressions

Elapsed Time: 0d:0h:0m:24s:345886200

Starting Equation:

```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

```
############################################
Attempt #: 1
Size: 151
Equation: o1 = ((((((((i6 + (i6 + i1)) * (i6 + i2)) * (i2 * i1)) + (((i6 + ((i6 + i1) * i2)) * i2) * i3)) * i4) + ((((i6 + i2) * (i6 + i1))


############################################
Attempt #: 2
Size: 77
Equation: o1 = (((((i1 * i5) * i3) + ((i1 * ((i2 * i6) * i3)) + ((i1 * (i3 * i1)) * i2))) + (i2 * (i4 * i1))) + ((((i2 * i1) * (i6 * (i3 * (:


############################################
Attempt #: 3
Size: 137
Equation: o1 = ((((i2 * i3) * i6) + ((((i2 * i4) * i1) + ((i3 * i1) + (((i2 * (i3 * i1)) + (i3 * (i5 * i1))) * i3))) * ((i4 * i2) * i1))) +


############################################
Attempt #: 4
Size: 69
Equation: o1 = ((i4 * (i1 * i2)) + (((((i6 * (i2 * i3)) * i3) + (i3 * (i1 * i2))) + ((i3 * i1) * i5)) + (((i4 * (i3 * ((i3 * i1) * i5))) + (:


############################################
Attempt #: 5
Size: 175
```

Final Equation:

```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

-------------Boolean Algebra Laws-------------

Logical

Annihilation: 1+A=1 [1]

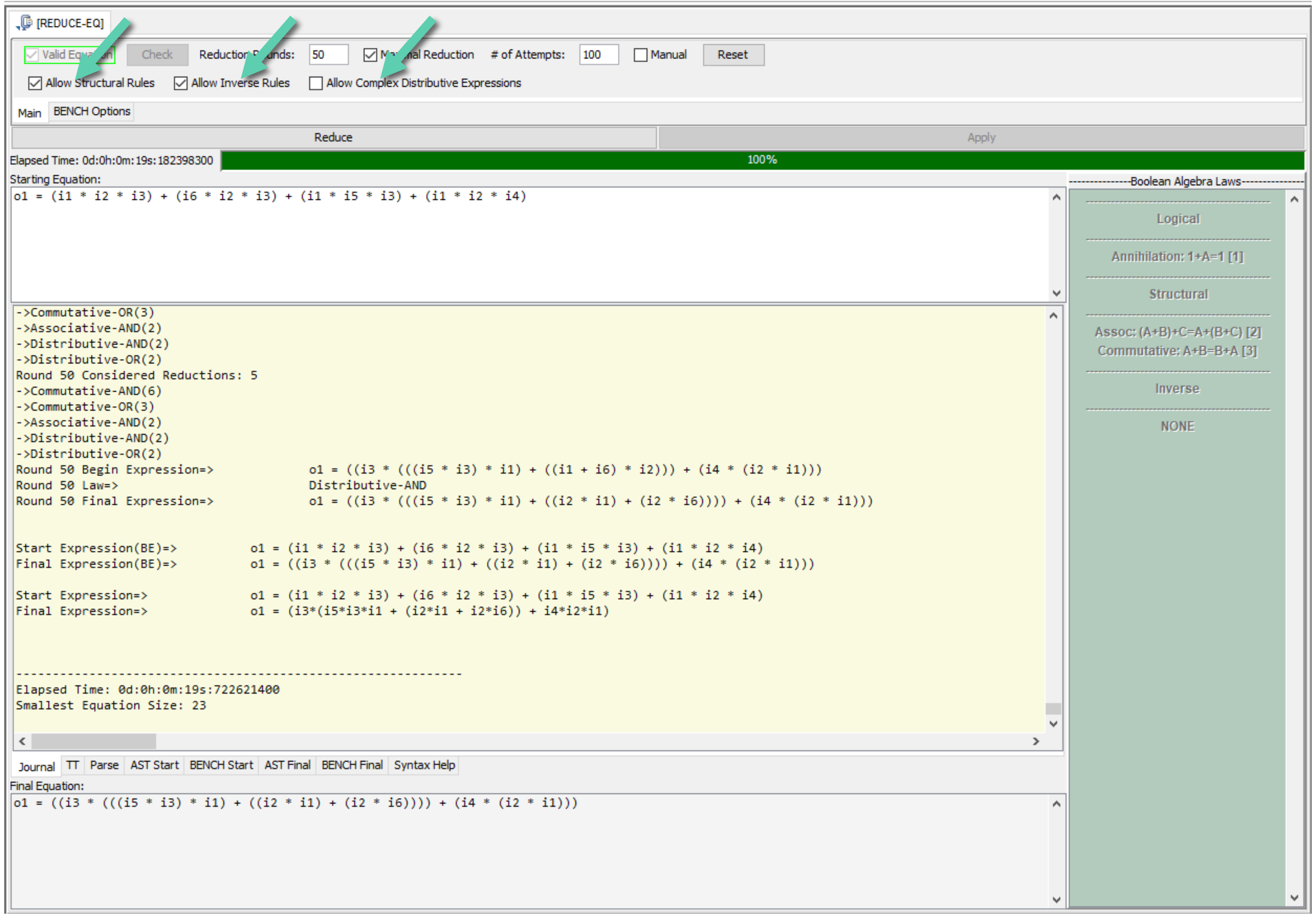
Structural

Assoc: (A+B)+C=A+(B+C) [2]
Commutative: A+B=B+A [3]

Inverse

NONE

- Allowing structured, inverse, and complex distributive expressions may open up alternative reduction sequences that may result in smaller sizes



[✓ Valid Equation] Reduction Rounds: 50 [✓ Maximal Reduction] # of Attempts: 100 ☐ Manual

[✓ Allow Structural Rules] [✓ Allow Inverse Rules] ☐ Allow Complex Distributive Expressions

Elapsed Time: 0d:0h:0m:19s:182398300

Starting Equation:

```
o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
```

```
->Commutative-OR(3)
->Associative-AND(2)
->Distributive-AND(2)
->Distributive-OR(2)
Round 50 Considered Reductions: 5
->Commutative-AND(6)
->Commutative-OR(3)
->Associative-AND(2)
->Distributive-AND(2)
->Distributive-OR(2)
Round 50 Begin Expression=>          o1 = ((i3 * (((i5 * i3) * i1) + ((i1 + i6) * i2))) + (i4 * (i2 * i1)))
Round 50 Law=>                       Distributive-AND
Round 50 Final Expression=>          o1 = ((i3 * (((i5 * i3) * i1) + ((i2 * i1) + (i2 * i6)))) + (i4 * (i2 * i1)))


Start Expression(BE)=>        o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression(BE)=>        o1 = ((i3 * (((i5 * i3) * i1) + ((i2 * i1) + (i2 * i6)))) + (i4 * (i2 * i1)))

Start Expression=>            o1 = (i1 * i2 * i3) + (i6 * i2 * i3) + (i1 * i5 * i3) + (i1 * i2 * i4)
Final Expression=>            o1 = (i3*(i5*i3*i1 + (i2*i1 + i2*i6)) + i4*i2*i1)



-------------------------------------------------------------
Elapsed Time: 0d:0h:0m:19s:722621400
Smallest Equation Size: 23
```

Final Equation:

```
o1 = ((i3 * (((i5 * i3) * i1) + ((i2 * i1) + (i2 * i6)))) + (i4 * (i2 * i1)))
```

------------Boolean Algebra Laws------------

Logical

Annihilation: 1+A=1 [1]

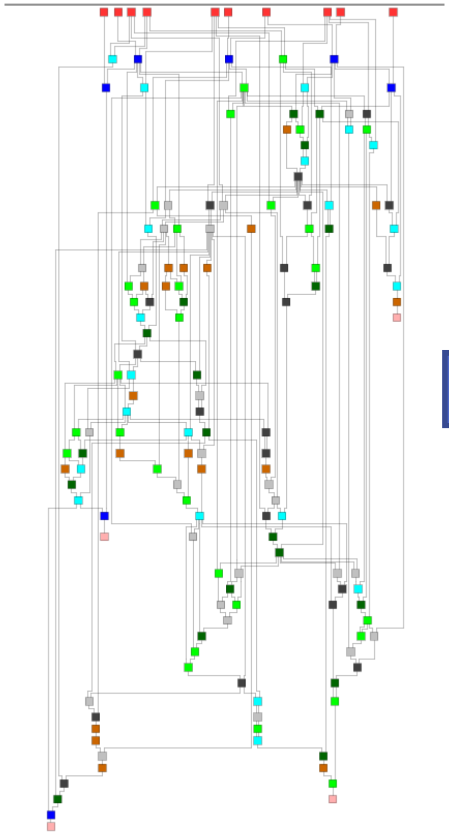
Structural

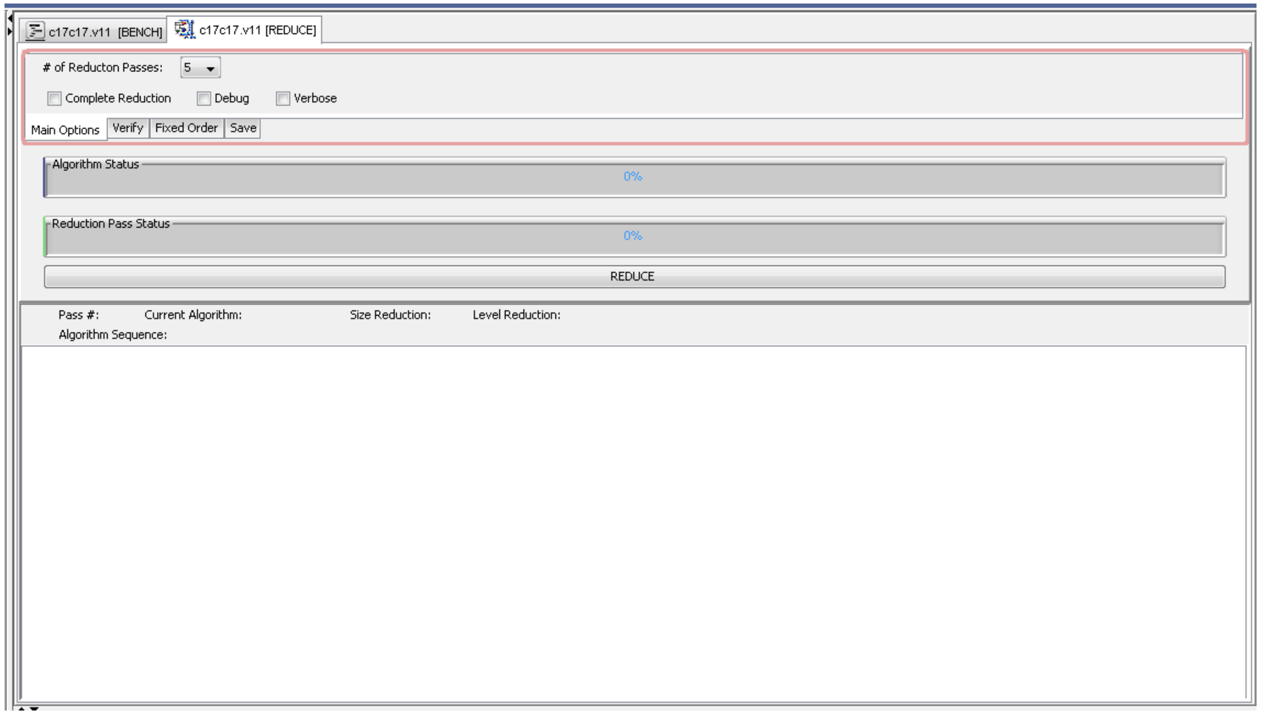
Assoc: (A+B)+C=A+(B+C) [2]
Commutative: A+B=B+A [3]

Inverse

NONE

Example: BENCH circuit selected in text panel





c17c17.v11 [BENCH] | c17c17.v11 [REDUCE]

# of Reducton Passes: 5

☐ Complete Reduction ☐ Debug ☐ Verbose

Main Options | Verify | Fixed Order | Save

Algorithm Status 0%

Reduction Pass Status 0%

REDUCE

Pass #: Current Algorithm: Size Reduction: Level Reduction:

Algorithm Sequence:

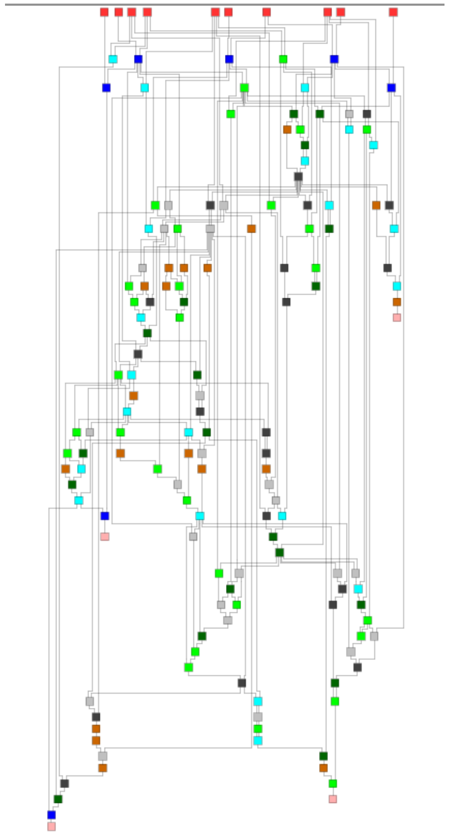
Pass: an application of all reduction algorithm, in some sequene

Complete Reduction: perform reduction rounds until two reduction rounds in a row no longer reduce the number of gates in the circuit

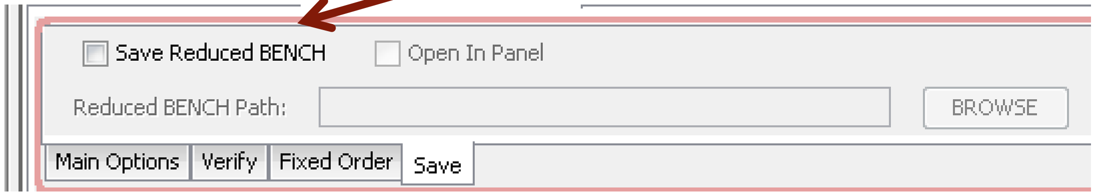
INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Circuit Reducer

Options:



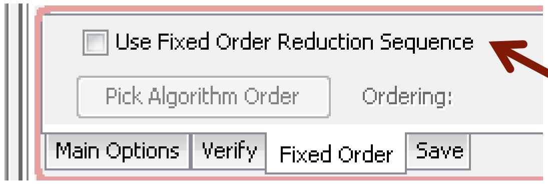
Save the reduced BENCH file and optionally open it as a text panel



☐ Save Reduced BENCH ☐ Open In Panel

Reduced BENCH Path: BROWSE

Main Options | Verify | Fixed Order | Save

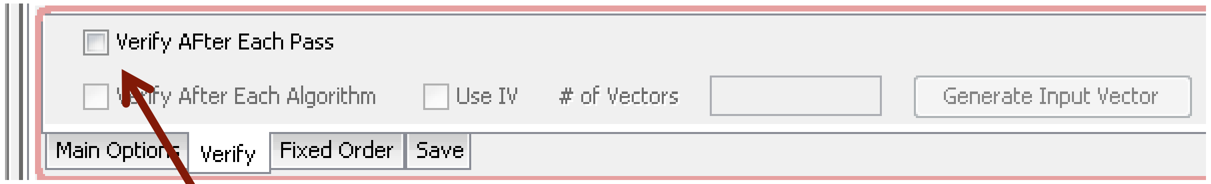


☐ Use Fixed Order Reduction Sequence

Pick Algorithm Order Ordering:

Main Options | Verify | Fixed Order | Save

Instead of the reducer choosing a random order of the reduction algorithms, you can specify a specific order instead



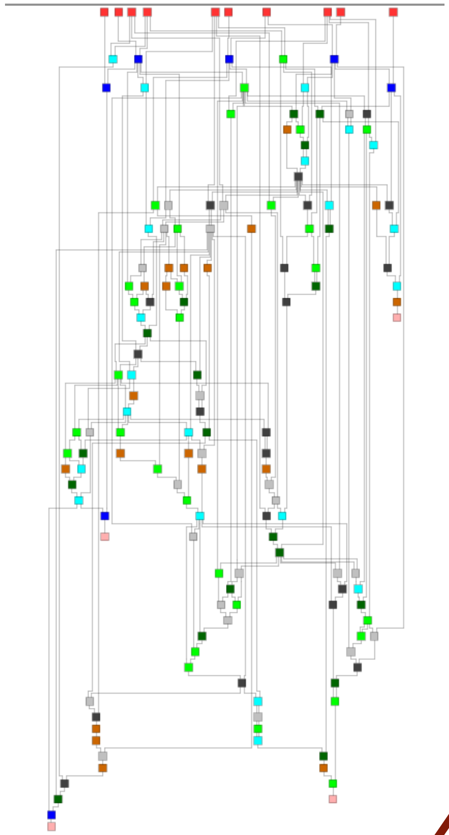
☐ Verify AFter Each Pass

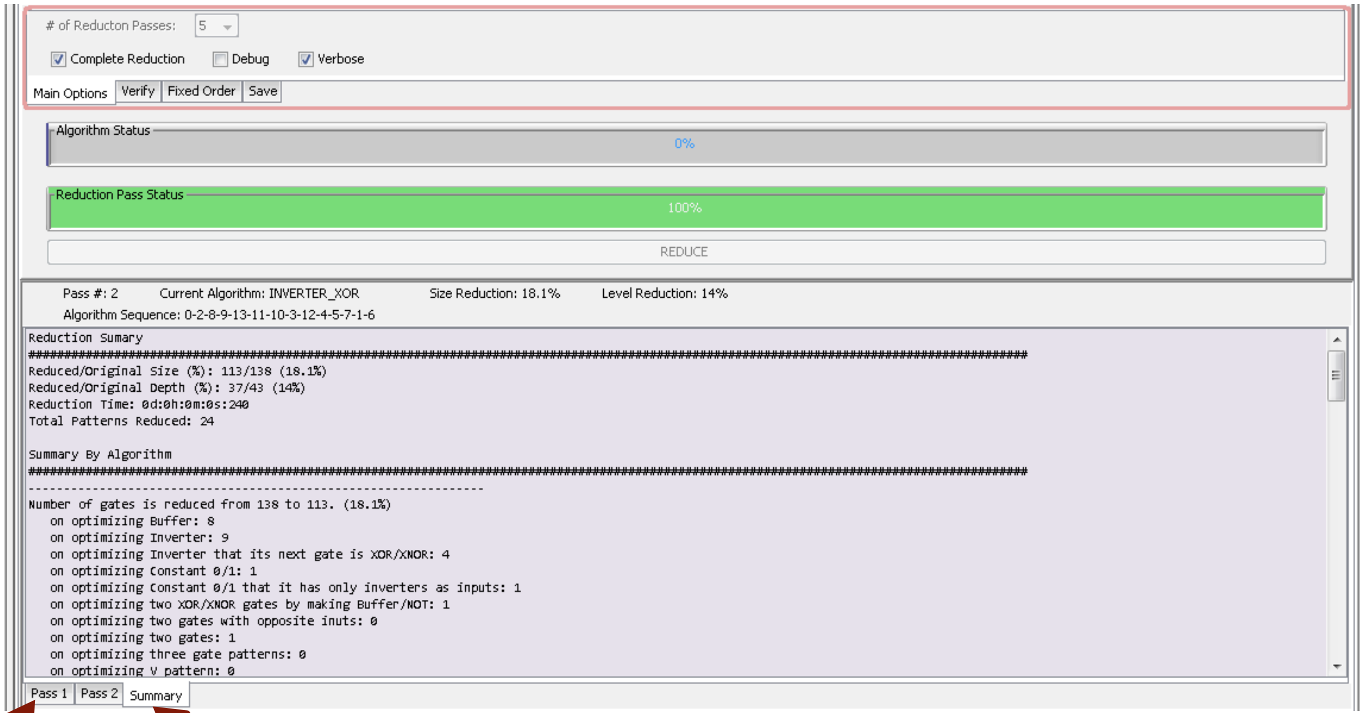
☐ Verify After Each Algorithm ☐ Use IV # of Vectors Generate Input Vector

Main Options | Verify | Fixed Order | Save

Verification options to check the reduced variant is semantically equivalent to the original circuit

After selecting REDUCE:





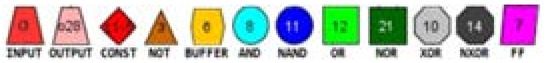
[✓ Complete Reduction] Debug [✓ Verbose]

```
Pass #: 2        Current Algorithm: INVERTER_XOR        Size Reduction: 18.1%        Level Reduction: 14%
Algorithm Sequence: 0-2-8-9-13-11-10-3-12-4-5-7-1-6
```

```
Reduction Sumary
##########################################################################################################################
Reduced/Original Size (%): 113/138 (18.1%)
Reduced/Original Depth (%): 37/43 (14%)
Reduction Time: 0d:0h:0m:0s:240
Total Patterns Reduced: 24

Summary By Algorithm
##########################################################################################################################
---------------------------------------------------------------
Number of gates is reduced from 138 to 113. (18.1%)
   on optimizing Buffer: 8
   on optimizing Inverter: 9
   on optimizing Inverter that its next gate is XOR/XNOR: 4
   on optimizing Constant 0/1: 1
   on optimizing Constant 0/1 that it has only inverters as inputs: 1
   on optimizing two XOR/XNOR gates by making Buffer/NOT: 1
   on optimizing two gates with opposite inuts: 0
   on optimizing two gates: 1
   on optimizing three gate patterns: 0
   on optimizing V pattern: 0
```

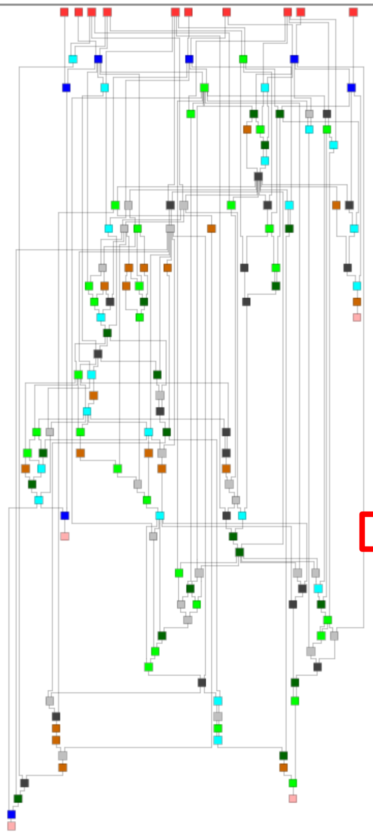
Overall summary

Per pass summary

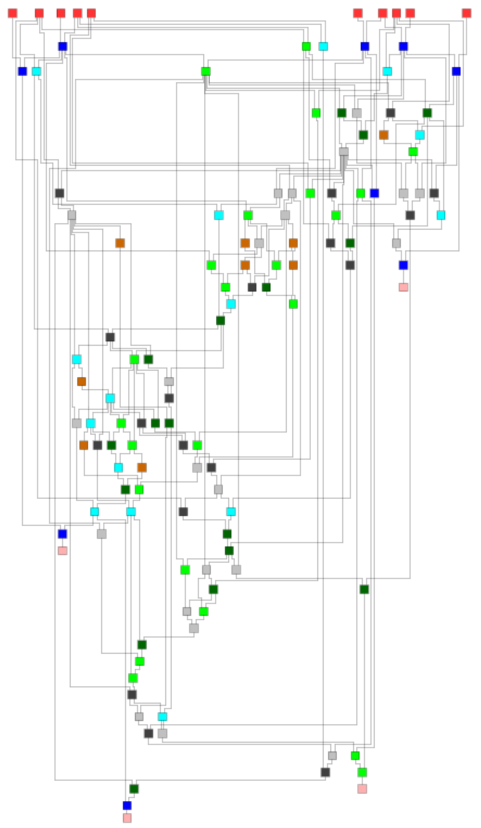


Circuit Reducer

After selecting REDUCE:



Number of inputs: 10
Number of outputs: 4
Number of constant1: 0
Number of constant0: 0
Intermediate gates: 138
ANDs: 21
ORs: 29
XORs: 23
NANDs: 7
NORs: 19
NXORs: 20
BUFFERs: 0
NOTs: 19
DFF: 0
JKFF: 0
TFF: 0
SRFF: 0
Circuit Depth: 43
Size of Largest Level: 10
Size of Largest Intermediate Level: 9
Maximum Fan-In: 2
Maximum Fan-Out: 10
Average Fan-In: 1.7
Average Fan-Out: 1.7



Number of inputs: 10
Number of outputs: 4
Number of constant1: 0
Number of constant0: 0
Intermediate gates: 113
ANDs: 15
ORs: 23
XORs: 22
NANDs: 9
NORs: 17
NXORs: 18
BUFFERs: 0
NOTs: 9
DFF: 0
JKFF: 0
TFF: 0
SRFF: 0
Circuit Depth: 37
Size of Largest Level: 10
Size of Largest Intermediate Level: 10
Maximum Fan-In: 2
Maximum Fan-Out: 10
Average Fan-In: 1.7
Average Fan-Out: 1.7

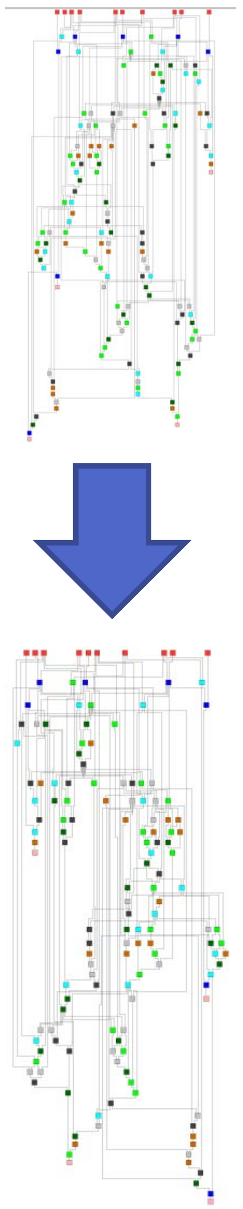
The currently implemented circuit reducer is based on pattern matching, with a view toward ***early*** implementations of the CORGI algorithm and the manner in which it accomplished polymorphic variation

In this example, the I/O space is tractably enumerable and normal logic synthesis would normally be used to reduce such a circuit to its smallest form: the benefits of pattern matching are most notable in larger circuits when standard synthesis techniques are not practical

INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

Structural and Shaped Reduction

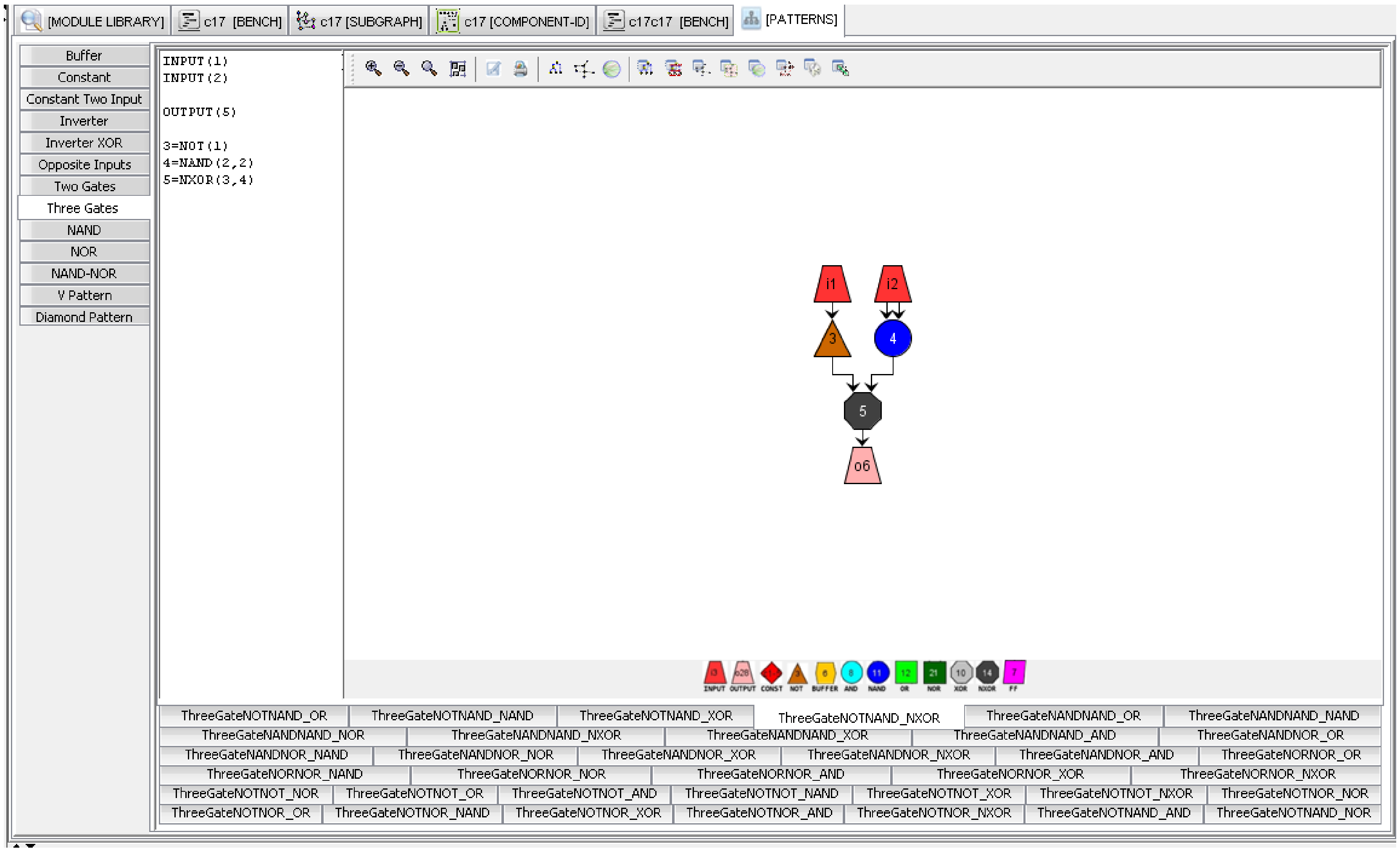
Any single pattern reduction algorithm can be applied to a circuit



INPUT
OUTPUT
CONST
NOT
BUFFER
AND
NAND
OR
NOR
XOR
NXOR
FF

```
Reduction Type: BUFFER

Total Patterns Reduced: 7
Total Time: 0d:0h:0m:0s:10
% Size Reduction: 5.07
% Level Reduction: 2.33

################################################################################

Buffer Gates Reduced: 0
Buffer Gate Pattern 1 Reduced: 4
Buffer Gate Pattern 2 Reduced: 3

################################################################################
Buffer Gates Time: 0d:0h:0m:0s:0
Buffer Gate Pattern 1 Time: 0d:0h:0m:0s:10
Buffer Gate Pattern 2 Time: 0d:0h:0m:0s:0

################################################################################

Original Circuit:
 [ Size = 138 ]
 [ Depth = 43 ]
 [ Avg Fan In = 1.7172 ]
 [ Avg Fan Out = 1.7172 ]
 [ Max Fan In = 2 ]
 [ Max Fan Out = 10 ]
 [ Max Nodes Per Level = 9 ]
 [ AND=21 OR=29 XOR=23 NAND=7 NOR=19 BUFFER=0 NOT=19 ]

################################################################################

Reduced Circuit:
 [ Size = 131 ]
 [ Depth = 42 ]
 [ Avg Fan In = 1.7035 ]
 [ Avg Fan Out = 1.7035 ]
 [ Max Fan In = 2 ]
 [ Max Fan Out = 10 ]
 [ Max Nodes Per Level = 9 ]
 [ AND=18 OR=25 XOR=23 NAND=7 NOR=19 BUFFER=0 NOT=19 ]
```

Pattern Viewer



```
INPUT(1)
INPUT(2)

OUTPUT(5)

3=NOT(1)
4=NAND(2,2)
5=NXOR(3,4)
```

ThreeGateNOTNAND_OR	ThreeGateNOTNAND_NAND	ThreeGateNOTNAND_XOR	ThreeGateNOTNAND_NXOR	ThreeGateNANDNAND_OR	ThreeGateNANDNAND_NAND
ThreeGateNANDNAND_NOR	ThreeGateNANDNAND_NXOR	ThreeGateNANDNAND_XOR	ThreeGateNANDNAND_AND	ThreeGateNANDNOR_OR	
ThreeGateNANDNOR_NAND	ThreeGateNANDNOR_NOR	ThreeGateNANDNOR_XOR	ThreeGateNANDNOR_NXOR	ThreeGateNANDNOR_AND	ThreeGateNORNOR_OR
ThreeGateNORNOR_NAND	ThreeGateNORNOR_NOR	ThreeGateNORNOR_AND	ThreeGateNORNOR_XOR	ThreeGateNORNOR_NXOR	
ThreeGateNOTNOT_NOR	ThreeGateNOTNOT_OR	ThreeGateNOTNOT_AND	ThreeGateNOTNOT_NAND	ThreeGateNOTNOT_XOR	ThreeGateNOTNOT_NXOR	ThreeGateNOTNOR_NOR
ThreeGateNOTNOR_OR	ThreeGateNOTNOR_NAND	ThreeGateNOTNOR_XOR	ThreeGateNOTNOR_AND	ThreeGateNOTNOR_NXOR	ThreeGateNOTNAND_AND	ThreeGateNOTNAND_NOR

The viewer shows the definition for all structural and shaped pattern circuits used in reduction algorithms

Random Circuits:

Random Circuit Generator

Random Equivalent Generator (Merged Signature)

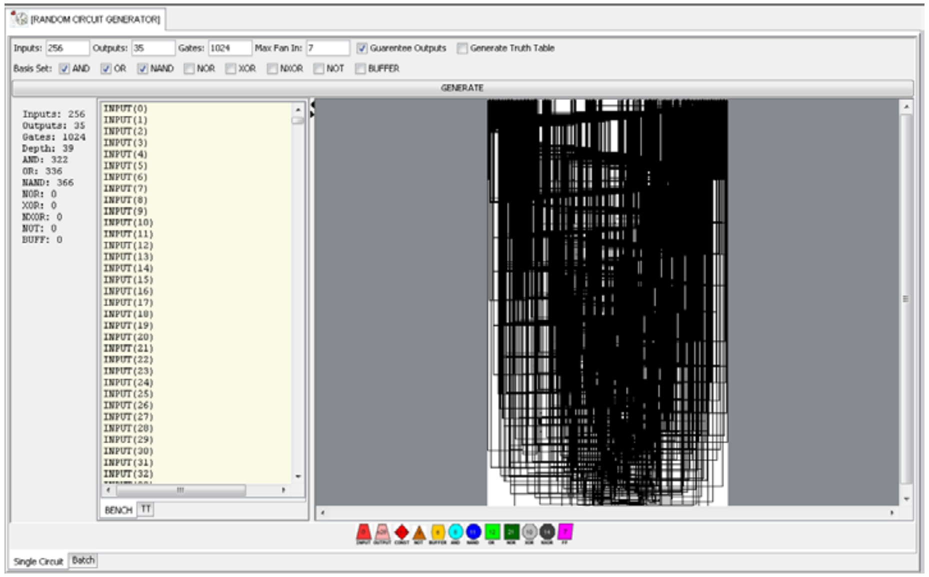
Random Equivalent Generator (Full Signature)

Random Circuit Generator

Single Circuit Mode

Walkthrough:

1) Choose inputs
2) Choose outputs
3) Choose size (# of gates)
4) Choose max fan-in
5) Guarantee # of outputs
6) Generate truth table (recommended for small I/O)
7) Pick basis set
8) Select GENERATE

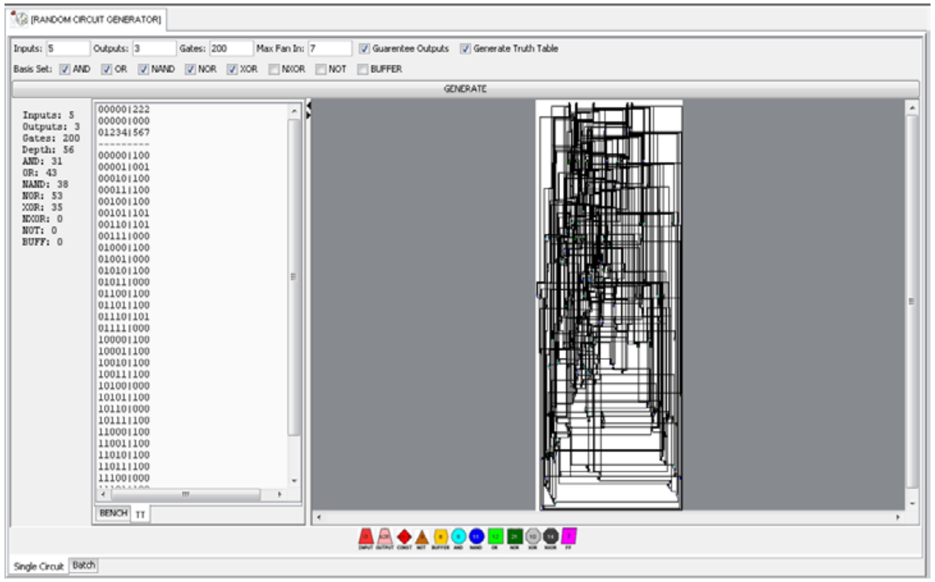


Inputs: 256 Outputs: 35 Gates: 1024 Max Fan In: 7 [✓ Guarentee Outputs] Generate Truth Table

Basis Set: [✓ AND] [✓ OR] [✓ NAND] NOR XOR NXOR NOT BUFFER

```
Inputs: 256
Outputs: 35
Gates: 1024
Depth: 39
AND: 322
OR: 336
NAND: 366
NOR: 0
XOR: 0
NXOR: 0
NOT: 0
BUFF: 0
```

```
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)
INPUT(10)
INPUT(11)
INPUT(12)
INPUT(13)
INPUT(14)
INPUT(15)
INPUT(16)
INPUT(17)
INPUT(18)
INPUT(19)
INPUT(20)
INPUT(21)
INPUT(22)
INPUT(23)
INPUT(24)
INPUT(25)
INPUT(26)
INPUT(27)
INPUT(28)
INPUT(29)
INPUT(30)
INPUT(31)
INPUT(32)
```



Inputs: 5 Outputs: 3 Gates: 200 Max Fan In: 7 [✓ Guarentee Outputs] [✓ Generate Truth Table]

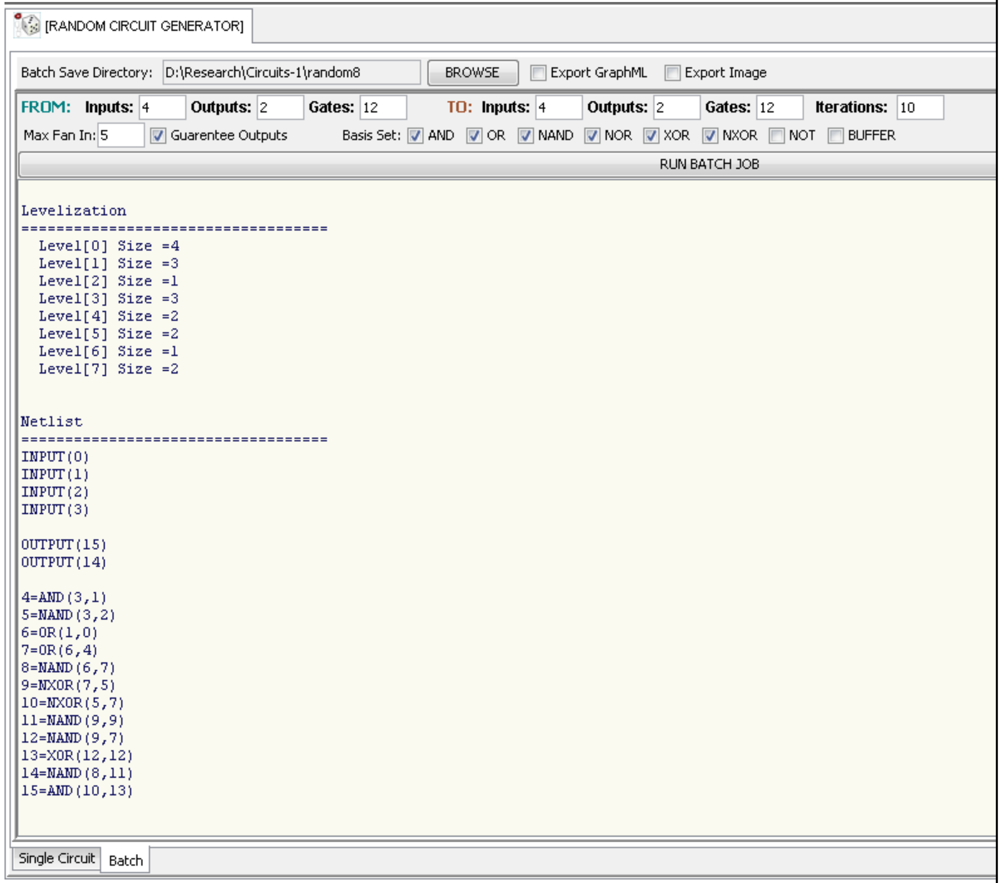
Basis Set: [✓ AND] [✓ OR] [✓ NAND] [✓ NOR] [✓ XOR] NXOR NOT BUFFER

```
Inputs: 5
Outputs: 3
Gates: 200
Depth: 56
AND: 31
OR: 43
NAND: 38
NOR: 53
XOR: 35
NXOR: 0
NOT: 0
BUFF: 0
```

```
00000|222
00000|000
01234|567
---------
00000|100
00001|001
00010|100
00011|100
00100|100
00101|101
00110|101
00111|000
01000|100
01001|000
01010|100
01011|000
01100|100
01101|100
01110|101
01111|000
10000|100
10001|100
10010|100
10011|100
10100|000
10101|100
10110|000
10111|100
11000|100
11001|100
11010|100
11011|100
11100|000
```

Random Circuit Generator

Batch Mode



Batch Save Directory: D:\Research\Circuits-1\random8 ☐ Export GraphML ☐ Export Image

FROM: Inputs: 4 Outputs: 2 Gates: 12 TO: Inputs: 4 Outputs: 2 Gates: 12 Iterations: 10

Max Fan In: 5 ✓ Guarentee Outputs Basis Set: ✓ AND ✓ OR ✓ NAND ✓ NOR ✓ XOR ✓ NXOR ☐ NOT ☐ BUFFER

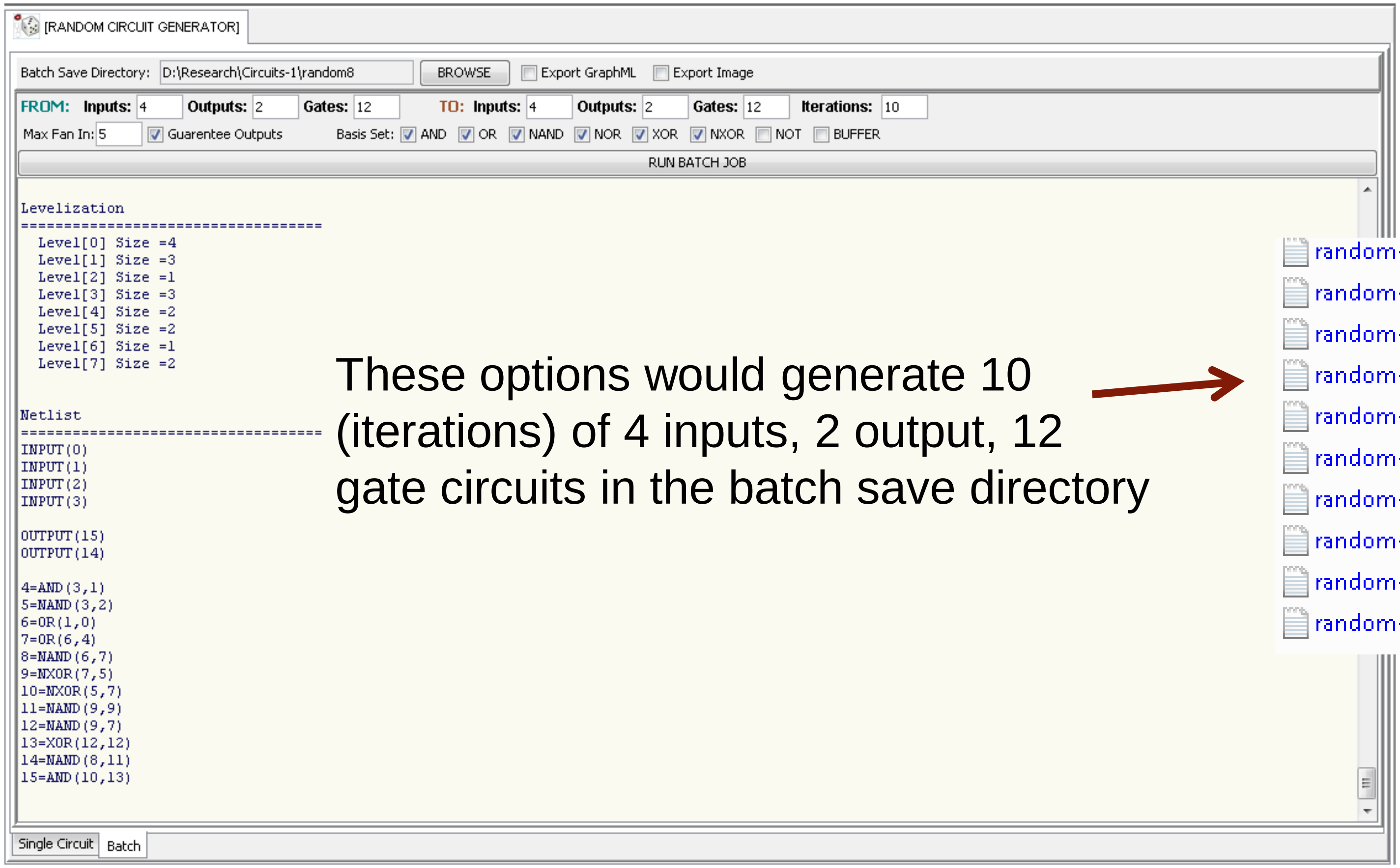
```
Levelization
===================================
  Level[0] Size =4
  Level[1] Size =3
  Level[2] Size =1
  Level[3] Size =3
  Level[4] Size =2
  Level[5] Size =2
  Level[6] Size =1
  Level[7] Size =2


Netlist
===================================
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(15)
OUTPUT(14)

4=AND(3,1)
5=NAND(3,2)
6=OR(1,0)
7=OR(6,4)
8=NAND(6,7)
9=NXOR(7,5)
10=NXOR(5,7)
11=NAND(9,9)
12=NAND(9,7)
13=XOR(12,12)
14=NAND(8,11)
15=AND(10,13)
```

Walkthrough:

1) Choose a save directory
2) Choose if you also want to save image or graphml files (in addition to BENCH)
3) Choose for loop constraints:
- Input Size (FROM/TO)
- Output Size (FROM/TO)
- Gate Size (FROM/TO)
- Iterations (how many of each random circuit should be generated)
4) Choose max fan-in
5) Guarantee # of outputs
6) Generate truth table (recommended for small I/O)
7) Pick basis set
8) Select GENERATE

Random Circuit Generator

Batch Mode



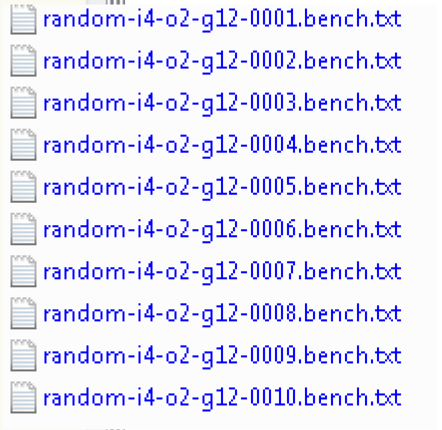
Batch Save Directory: D:\Research\Circuits-1\random8 [ ] Export GraphML [ ] Export Image

FROM: Inputs: 4 Outputs: 2 Gates: 12 TO: Inputs: 4 Outputs: 2 Gates: 12 Iterations: 10

Max Fan In: 5 [✓] Guarentee Outputs Basis Set: [✓] AND [✓] OR [✓] NAND [✓] NOR [✓] XOR [✓] NXOR [ ] NOT [ ] BUFFER

```
Levelization
===================================
  Level[0] Size =4
  Level[1] Size =3
  Level[2] Size =1
  Level[3] Size =3
  Level[4] Size =2
  Level[5] Size =2
  Level[6] Size =1
  Level[7] Size =2


Netlist
===================================
INPUT(0)
INPUT(1)
INPUT(2)
INPUT(3)

OUTPUT(15)
OUTPUT(14)

4=AND(3,1)
5=NAND(3,2)
6=OR(1,0)
7=OR(6,4)
8=NAND(6,7)
9=NXOR(7,5)
10=NXOR(5,7)
11=NAND(9,9)
12=NAND(9,7)
13=XOR(12,12)
14=NAND(8,11)
15=AND(10,13)
```

These options would generate 10 (iterations) of 4 inputs, 2 output, 12 gate circuits in the batch save directory

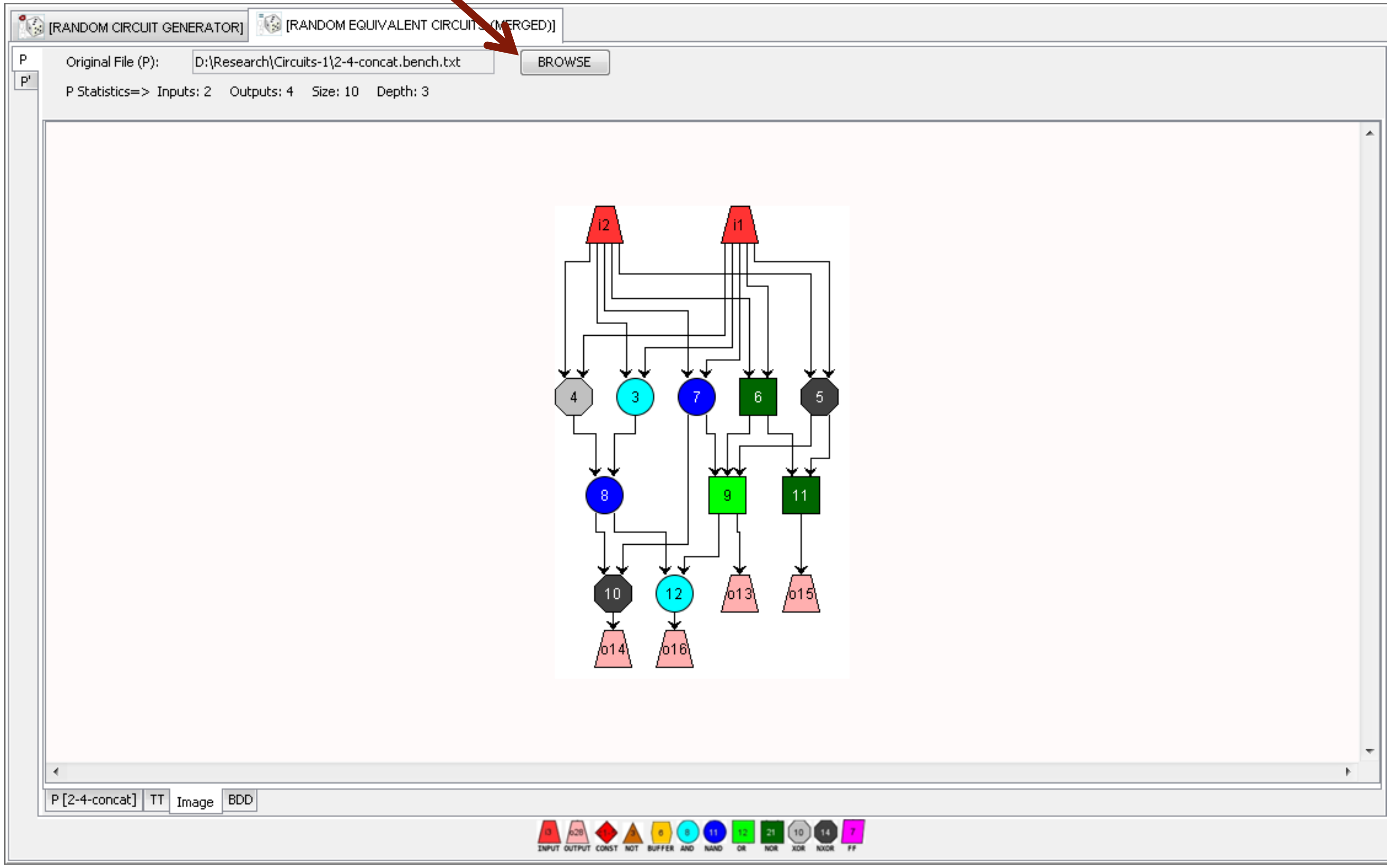


random-i4-o2-g12-0001.bench.txt
random-i4-o2-g12-0002.bench.txt
random-i4-o2-g12-0003.bench.txt
random-i4-o2-g12-0004.bench.txt
random-i4-o2-g12-0005.bench.txt
random-i4-o2-g12-0006.bench.txt
random-i4-o2-g12-0007.bench.txt
random-i4-o2-g12-0008.bench.txt
random-i4-o2-g12-0009.bench.txt
random-i4-o2-g12-0010.bench.txt

Random Equivalent Circuit (Merged Signature)

This option does not require a selected BENCH circuit to be loaded first

Load an original circuit first

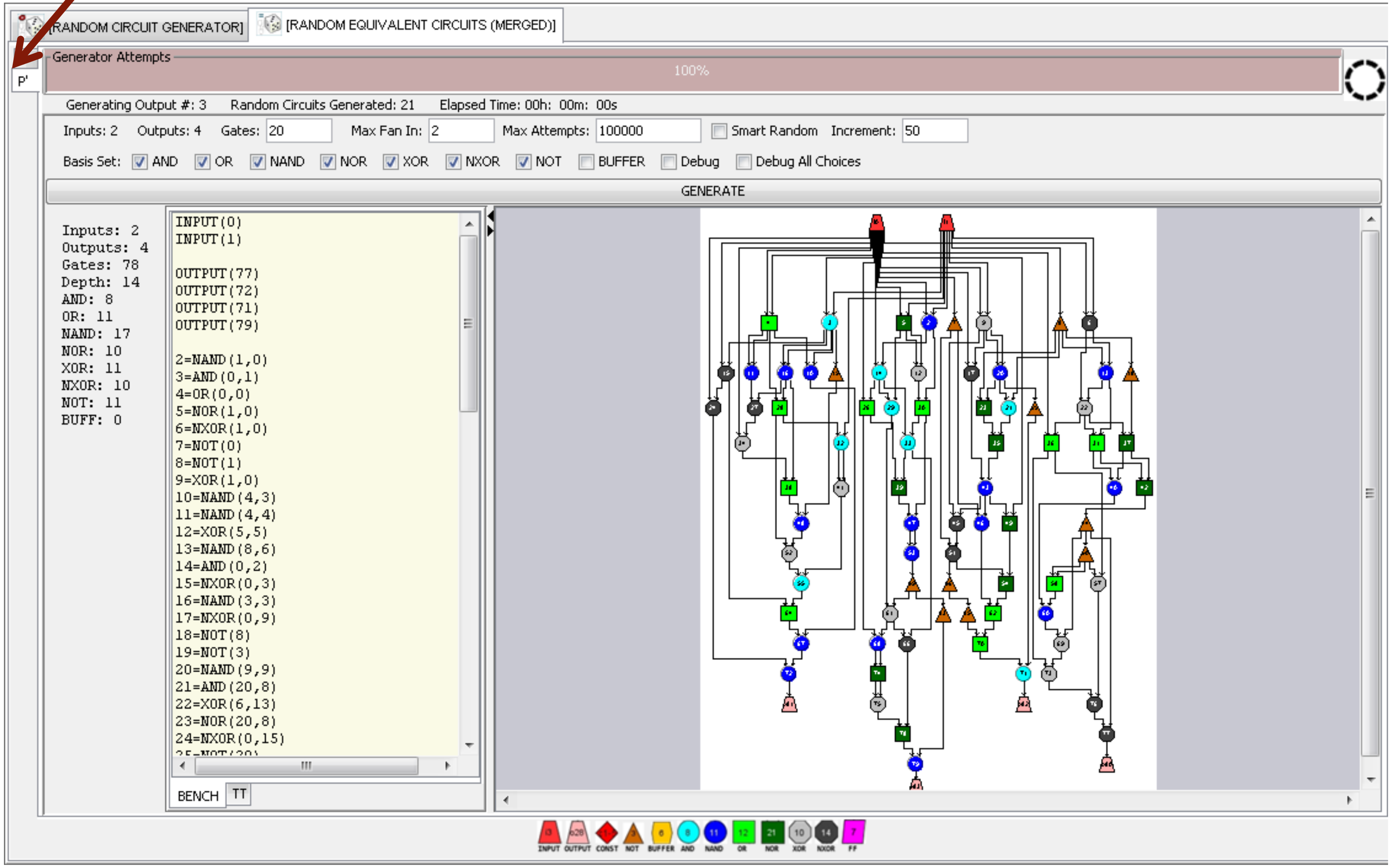

[RANDOM CIRCUIT GENERATOR] [RANDOM EQUIVALENT CIRCUITS (MERGED)]

Original File (P): D:\Research\Circuits-1\2-4-concat.bench.txt BROWSE

P Statistics=> Inputs: 2 Outputs: 4 Size: 10 Depth: 3

Random Equivalent Circuit (Merged Signature)

Select P’ tab, choose generation options, and select generate



Generator Attempts

Generating Output #: 3 Random Circuits Generated: 21 Elapsed Time: 00h: 00m: 00s

Inputs: 2 Outputs: 4 Gates: 20 Max Fan In: 2 Max Attempts: 100000 Smart Random Increment: 50

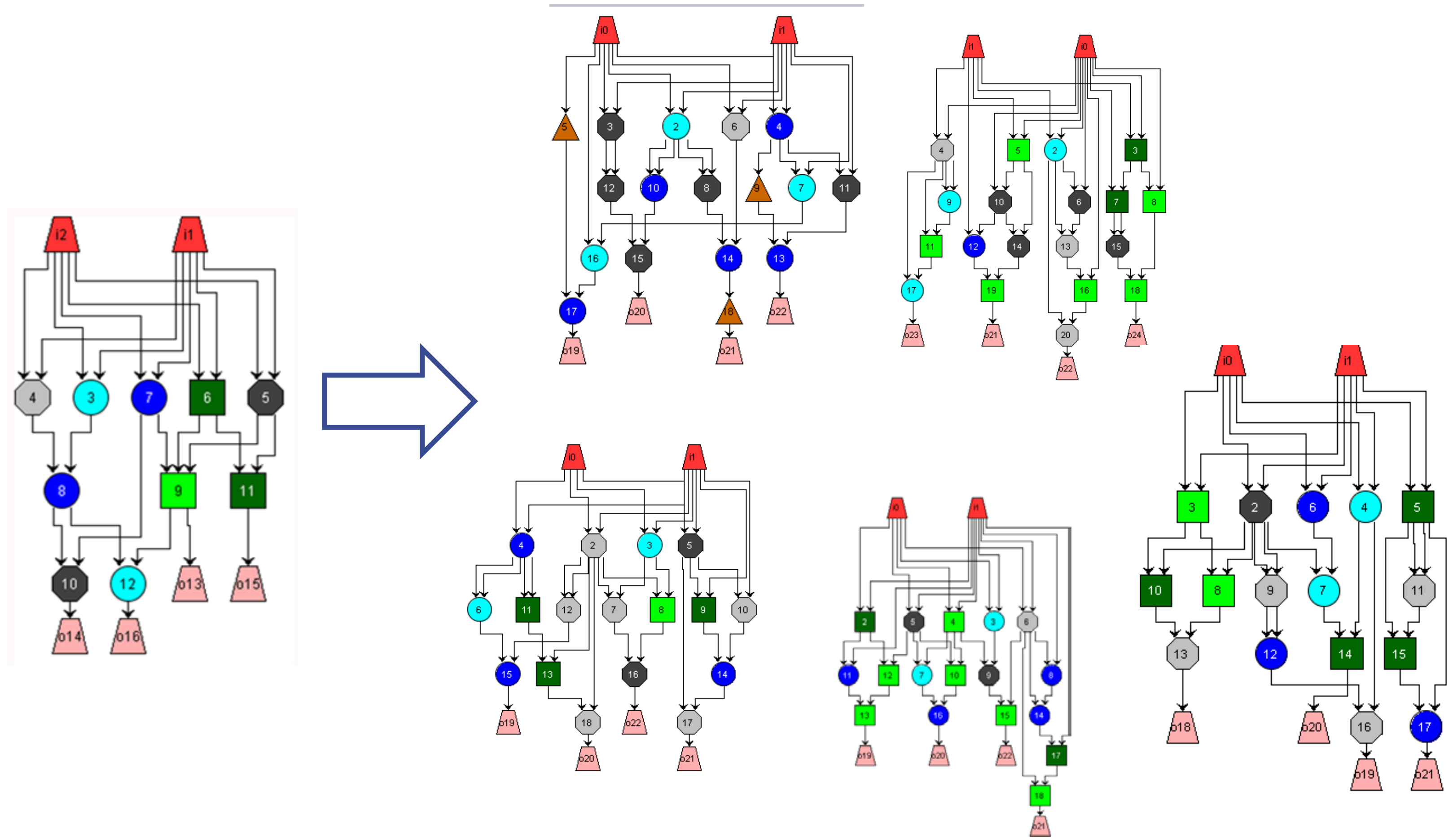
Basis Set: [✓ AND] [✓ OR] [✓ NAND] [✓ NOR] [✓ XOR] [✓ NXOR] [✓ NOT] BUFFER Debug Debug All Choices

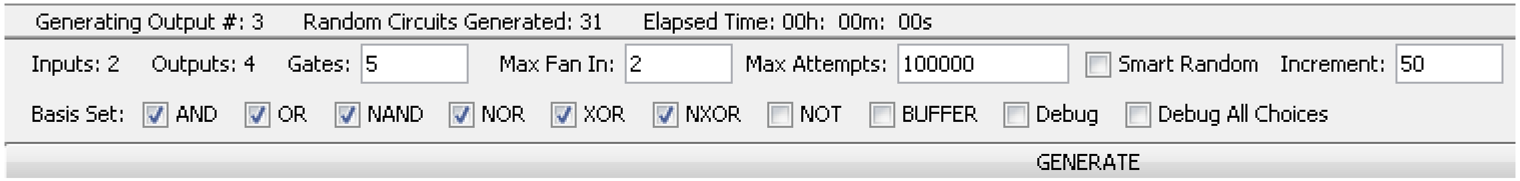
```
Inputs: 2
Outputs: 4
Gates: 78
Depth: 14
AND: 8
OR: 11
NAND: 17
NOR: 10
XOR: 11
NXOR: 10
NOT: 11
BUFF: 0
```

```
INPUT(0)
INPUT(1)

OUTPUT(77)
OUTPUT(72)
OUTPUT(71)
OUTPUT(79)

2=NAND(1,0)
3=AND(0,1)
4=OR(0,0)
5=NOR(1,0)
6=NXOR(1,0)
7=NOT(0)
8=NOT(1)
9=XOR(1,0)
10=NAND(4,3)
11=NAND(4,4)
12=XOR(5,5)
13=NAND(8,6)
14=AND(0,2)
15=NXOR(0,3)
16=NAND(3,3)
17=NXOR(0,9)
18=NOT(8)
19=NOT(3)
20=NAND(9,9)
21=AND(20,8)
22=XOR(6,13)
23=NOR(20,8)
24=NXOR(0,15)
25=NOT(20)
```

Random Equivalent Circuit (Merged Signature)

Continuing to hit GENERATE will create another variant:





Generating Output #: 3   Random Circuits Generated: 31   Elapsed Time: 00h: 00m: 00s

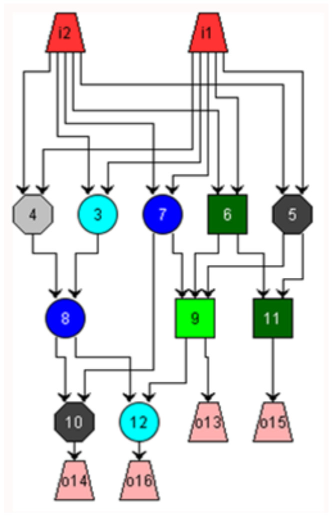
Inputs: 2   Outputs: 4   Gates: 5   Max Fan In: 2   Max Attempts: 100000   ☐ Smart Random   Increment: 50

Basis Set: ☑ AND ☑ OR ☑ NAND ☑ NOR ☑ XOR ☑ NXOR ☐ NOT ☐ BUFFER ☐ Debug ☐ Debug All Choices

GENERATE

Random Equivalent Circuit (Merged Signature)

Merged Signature Random Circuits:
These circuits are created by generating an equivalent random circuit for each OUTPUT of the original circuit, and then MERGING those individual circuits into a single circuit



```
00|1122
01|8901
-------
00|1101
01|1111
10|1111
11|1001
```

random1
random2
random3
random4

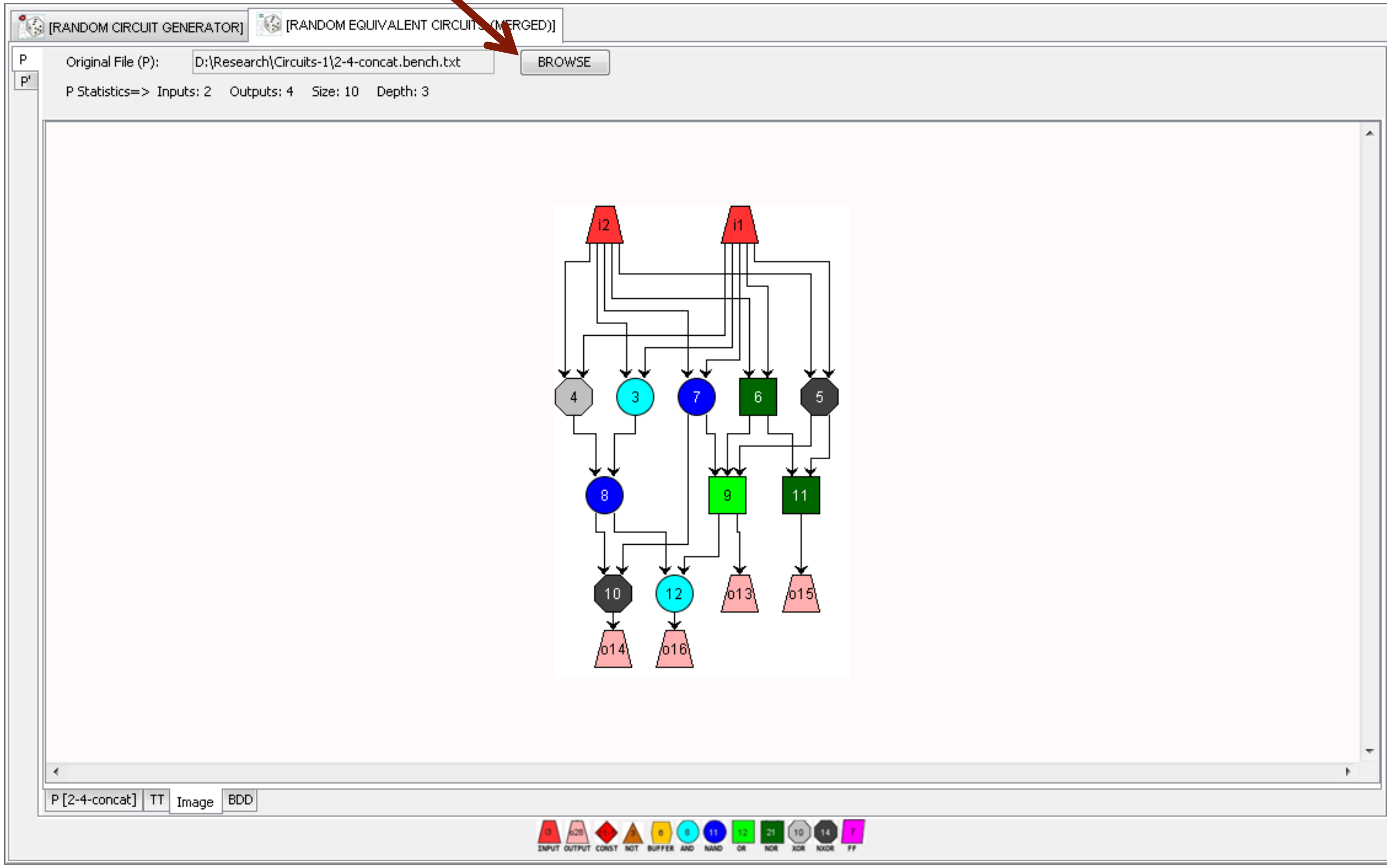
$\left.\begin{array}{l}\text{random1}\\\text{random2}\\\text{random3}\\\text{random4}\end{array}\right\}$ random circuit

2 input / 1 output / X gate circuits

Random Equivalent Circuit (Full Signature)

This option does not require a selected BENCH circuit to be loaded first

Load an original circuit first, then switch to P' tab



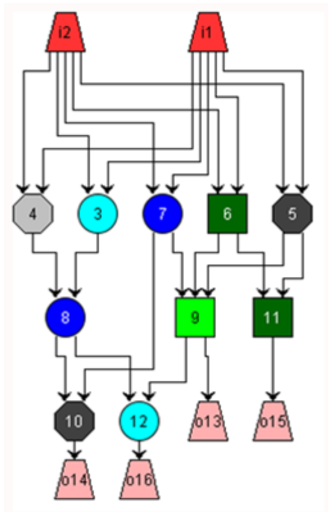
Original File (P): D:\Research\Circuits-1\2-4-concat.bench.txt

P Statistics=> Inputs: 2 Outputs: 4 Size: 10 Depth: 3

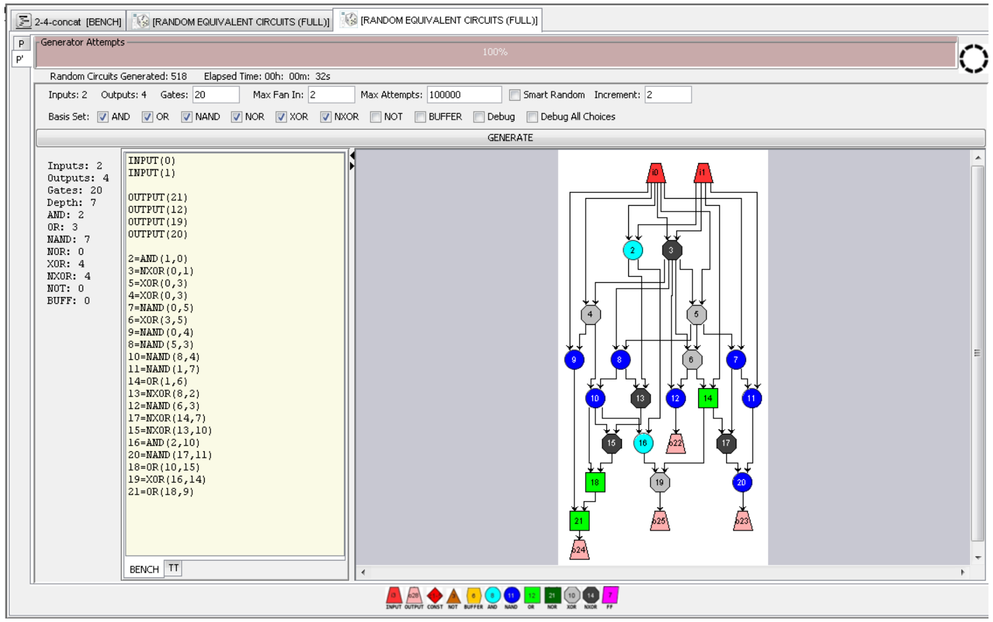
Random Equivalent Circuit (Full Signature)

Full Signature Random Circuits:
These circuits are created by generating random circuits that match the entire input/output size of the original circuit. Generation continues until a circuit with a matching signature is generated OR max generation attempts are reached.

20 gate variant



```
00|1122
01|8901
-------
00|1101
01|1111
10|1111
11|1001
```


2-4-concat [BENCH]
[RANDOM EQUIVALENT CIRCUITS (FULL)]
[RANDOM EQUIVALENT CIRCUITS (FULL)]
P
P'
Generator Attempts
100%
GENERATE
BENCH
TT

Random Circuits Generated: 518 Elapsed Time: 00h: 00m: 32s

Inputs: 2 Outputs: 4 Gates: 20 Max Fan In: 2 Max Attempts: 100000 ☐ Smart Random Increment: 2

Basis Set: ☑ AND ☑ OR ☑ NAND ☑ NOR ☑ XOR ☑ NXOR ☐ NOT ☐ BUFFER ☐ Debug ☐ Debug All Choices

```
Inputs: 2
Outputs: 4
Gates: 20
Depth: 7
AND: 2
OR: 3
NAND: 7
NOR: 0
XOR: 4
NXOR: 4
NOT: 0
BUFF: 0
```

```
INPUT(0)
INPUT(1)

OUTPUT(21)
OUTPUT(12)
OUTPUT(19)
OUTPUT(20)

2=AND(1,0)
3=NXOR(0,1)
5=XOR(0,3)
4=XOR(0,3)
7=NAND(0,5)
6=XOR(3,5)
9=NAND(0,4)
8=NAND(5,3)
10=NAND(8,4)
11=NAND(1,7)
14=OR(1,6)
13=NXOR(8,2)
12=NAND(6,3)
17=NXOR(14,7)
15=NXOR(13,10)
16=AND(2,10)
20=NAND(17,11)
18=OR(10,15)
19=XOR(16,14)
21=OR(18,9)
```

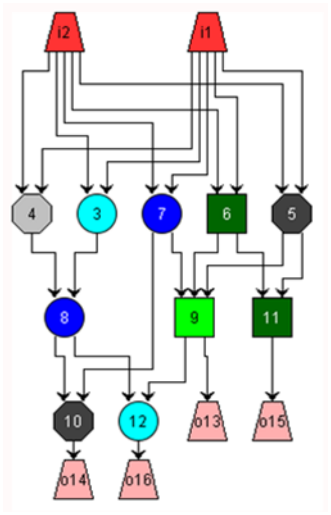
Full Signature = 1111111001101111

In general, size has to be adjusted for a reasonable possibility of generating the maximum range of signatures

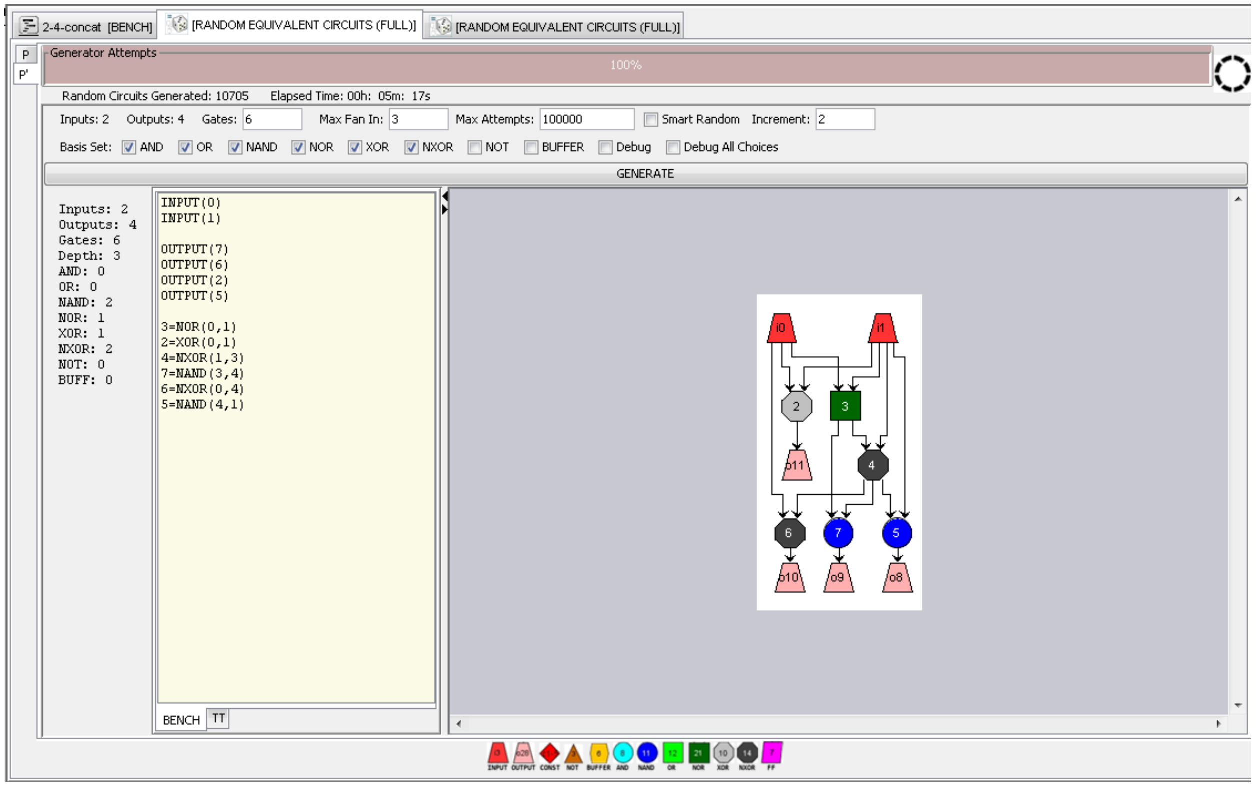
Random Equivalent Circuit (Full Signature)

Full Signature = 1111111001101111

6 gate variant



```
00|1122
01|8901
--------
00|1101
01|1111
10|1111
11|1001
```



2-4-concat [BENCH] | [RANDOM EQUIVALENT CIRCUITS (FULL)] | [RANDOM EQUIVALENT CIRCUITS (FULL)]

Generator Attempts 100%

Random Circuits Generated: 10705 Elapsed Time: 00h: 05m: 17s

Inputs: 2 Outputs: 4 Gates: 6 Max Fan In: 3 Max Attempts: 100000 Smart Random Increment: 2

Basis Set: [✓ AND] [✓ OR] [✓ NAND] [✓ NOR] [✓ XOR] [✓ NXOR] NOT BUFFER Debug Debug All Choices

GENERATE

```
Inputs: 2
Outputs: 4
Gates: 6
Depth: 3
AND: 0
OR: 0
NAND: 2
NOR: 1
XOR: 1
NXOR: 2
NOT: 0
BUFF: 0
```

```
INPUT(0)
INPUT(1)

OUTPUT(7)
OUTPUT(6)
OUTPUT(2)
OUTPUT(5)

3=NOR(0,1)
2=XOR(0,1)
4=NXOR(1,3)
7=NAND(3,4)
6=NXOR(0,4)
5=NAND(4,1)
```

Tradeoff with merged vs full signature is that it may take longer or max generation attempts may be reached using the full signature approach: however, the merged signature approach generates larger circuits

Use Case: Loading and Convert PLA File

1. File->Open->PLA

- PLA formats come from supported files used by the original SIS system

1. PLA format - Programmable Logic Array

Given a circuit, how do we describe it in the PLA format?

Consider the following circuit.

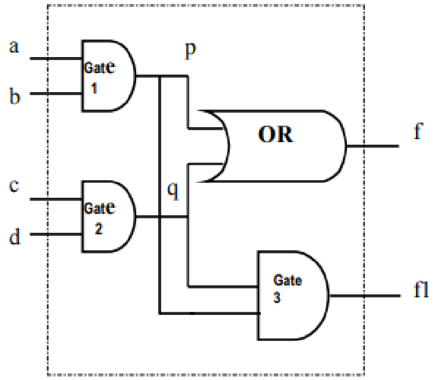


Fig. 1

In the above circuit,

1. Note the number of inputs, it is 4. (a,b,c,d) **{Specified by ".i"}**
2. Note the number of outputs, it is 2. (f, f1) **{Specified by ".o"}**

So, in the pla file we write

```
.i 4
.o 2
```

←PLA file

Naming the inputs and outputs

3. We define the names of the wires in the input.

   For the above circuit it is a, b, c, d

4. We define the names of wires in the output.

   For the above circuit it is f, f1

```
.i 4
.o 2
.ilb a b c d
.ob f f1
```

←PLA file

Giving the Truth Table

5. After specifying the inputs and outputs, we specify the truth table of the circuit. The number of terms in the truth table is represented by ".p" in the pla file.

A	B	C	D	F	F1
1	1	-	-	1	0
-	-	1	1	1	0
1	1	1	1	1	1

For the above truth table, the last 4 lines in the following snapshot have been added:

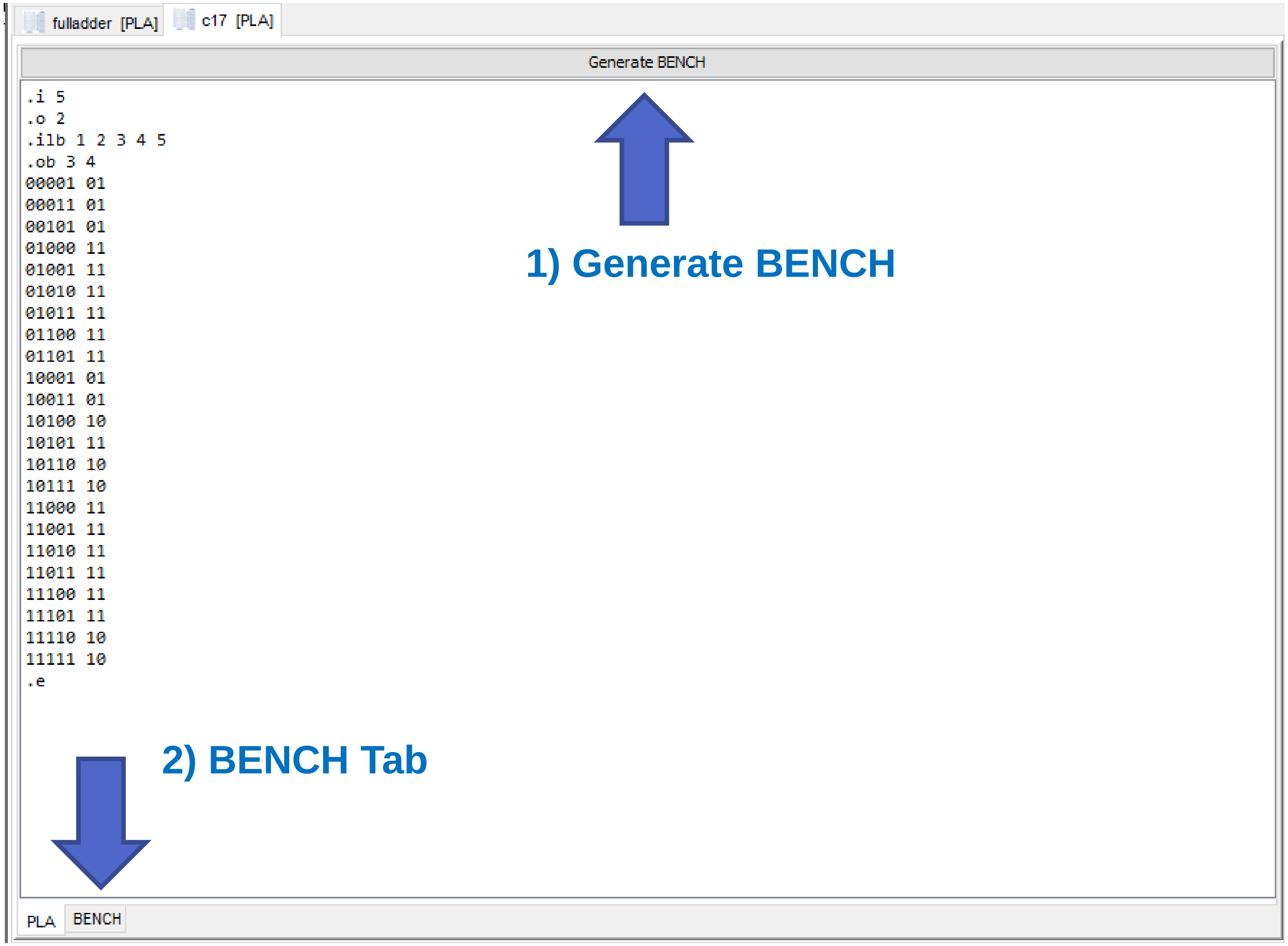
```
.i 4
.o 2
.ilb a b c d
.ob f f1
.p 3
 11-- 10
 --11 10
1111 11
```

←PLA file

https://ptolemy.berkeley.edu/projects/embedded/Alumni/pchong/sis /

Use Case: Loading and Convert PLA File

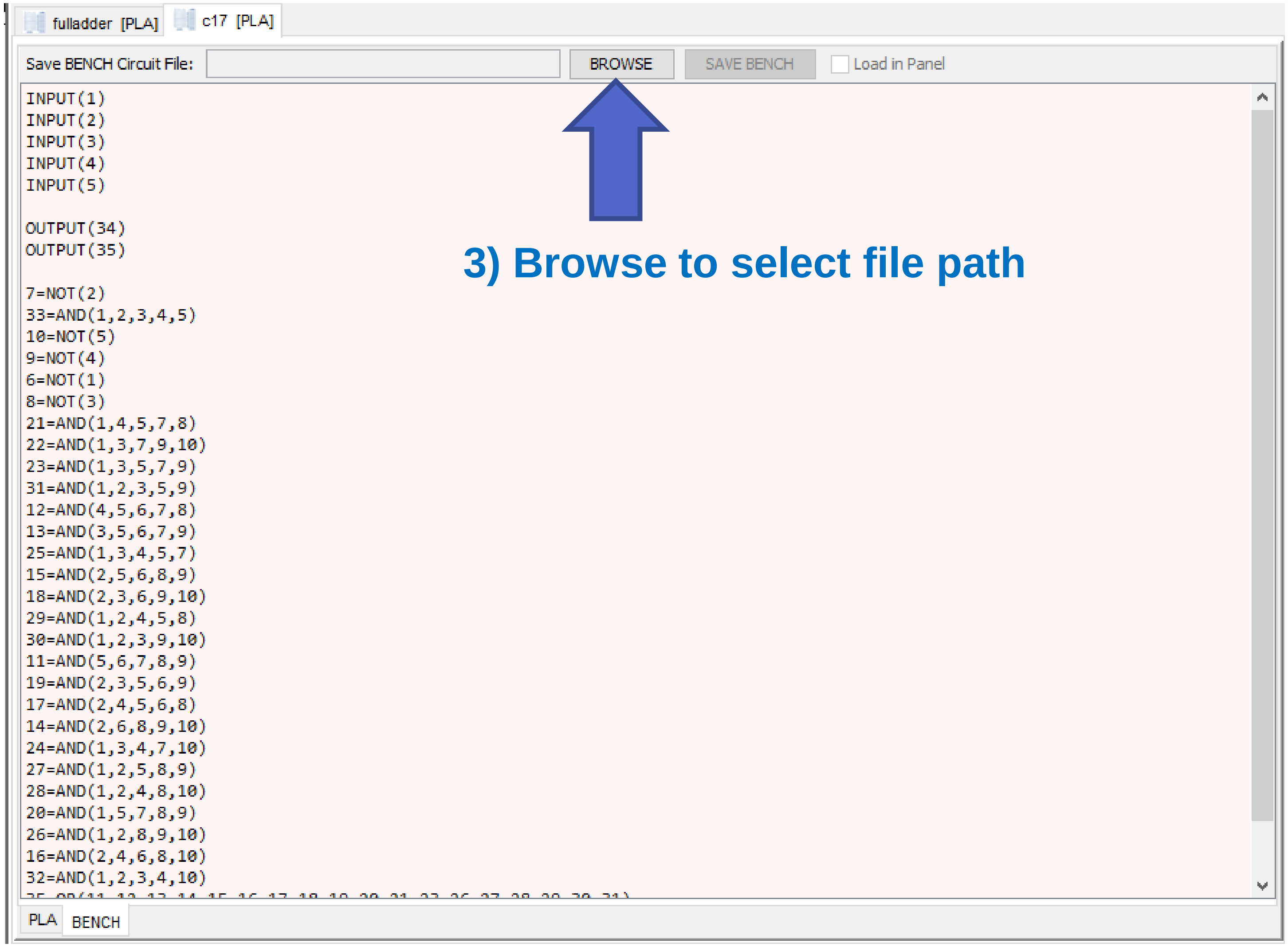
- Example PLA file (.pla)



```
.i 5
.o 2
.ilb 1 2 3 4 5
.ob 3 4
00001 01
00011 01
00101 01
01000 11
01001 11
01010 11
01011 11
01100 11
01101 11
10001 01
10011 01
10100 10
10101 11
10110 10
10111 10
11000 11
11001 11
11010 11
11011 11
11100 11
11101 11
11110 10
11111 10
.e
```

Use Case: Loading and Convert PLA File

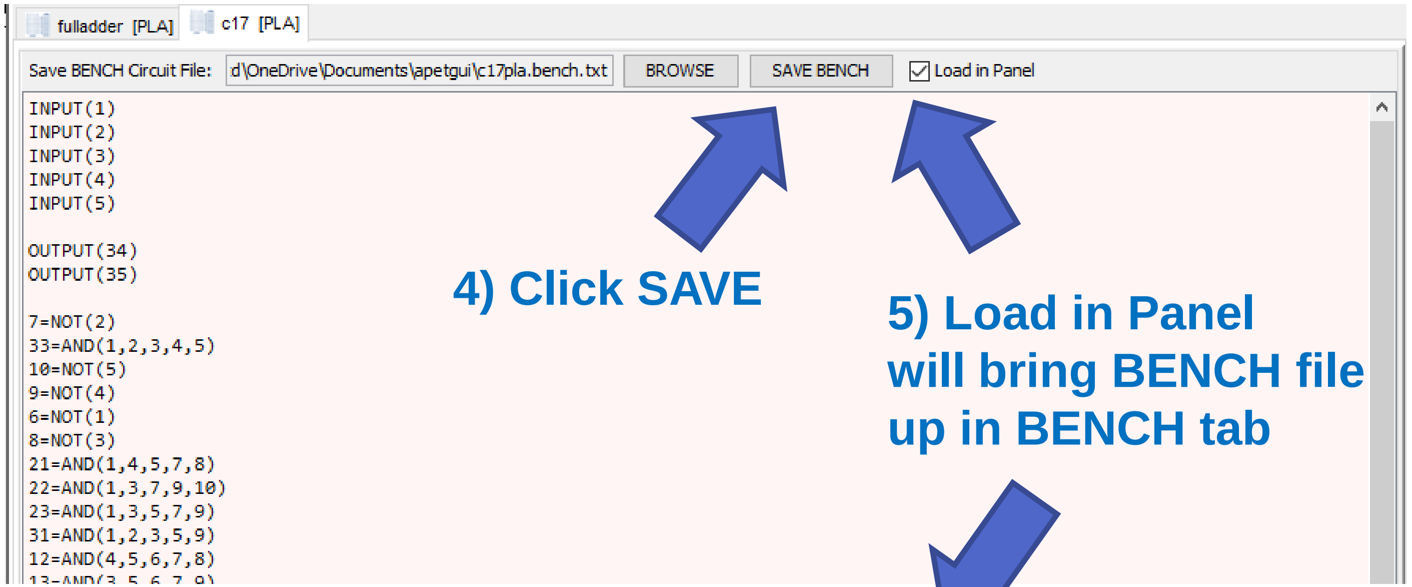
- Example PLA file (.pla)



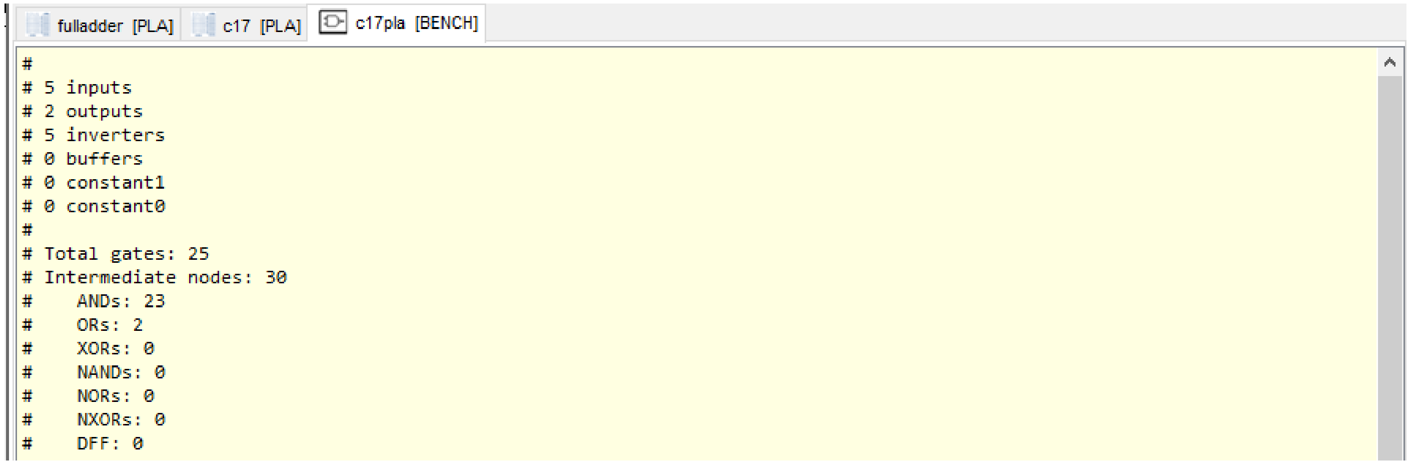
```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(34)
OUTPUT(35)

7=NOT(2)
33=AND(1,2,3,4,5)
10=NOT(5)
9=NOT(4)
6=NOT(1)
8=NOT(3)
21=AND(1,4,5,7,8)
22=AND(1,3,7,9,10)
23=AND(1,3,5,7,9)
31=AND(1,2,3,5,9)
12=AND(4,5,6,7,8)
13=AND(3,5,6,7,9)
25=AND(1,3,4,5,7)
15=AND(2,5,6,8,9)
18=AND(2,3,6,9,10)
29=AND(1,2,4,5,8)
30=AND(1,2,3,9,10)
11=AND(5,6,7,8,9)
19=AND(2,3,5,6,9)
17=AND(2,4,5,6,8)
14=AND(2,6,8,9,10)
24=AND(1,3,4,7,10)
27=AND(1,2,5,8,9)
28=AND(1,2,4,8,10)
20=AND(1,5,7,8,9)
26=AND(1,2,8,9,10)
16=AND(2,4,6,8,10)
32=AND(1,2,3,4,10)
35=OR(11,12,13,14,15,16,17,18,19,20,21,23,26,27,28,29,30,31)
```

Use Case: Loading and Convert PLA File

- Example PLA file (.pla)


fulladder [PLA]
c17 [PLA]
Save BENCH Circuit File:
d\OneDrive\Documents\apetgui\c17pla.bench.txt
BROWSE
SAVE BENCH
Load in Panel
4) Click SAVE
5) Load in Panel will bring BENCH file up in BENCH tab

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)

OUTPUT(34)
OUTPUT(35)

7=NOT(2)
33=AND(1,2,3,4,5)
10=NOT(5)
9=NOT(4)
6=NOT(1)
8=NOT(3)
21=AND(1,4,5,7,8)
22=AND(1,3,7,9,10)
23=AND(1,3,5,7,9)
31=AND(1,2,3,5,9)
12=AND(4,5,6,7,8)
```

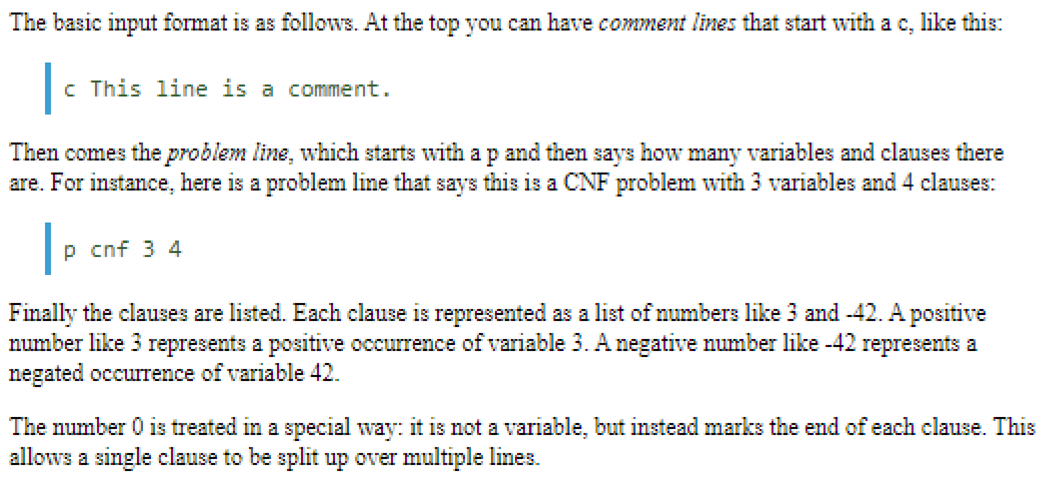


```
#
# 5 inputs
# 2 outputs
# 5 inverters
# 0 buffers
# 0 constant1
# 0 constant0
#
# Total gates: 25
# Intermediate nodes: 30
#    ANDs: 23
#    ORs: 2
#    XORs: 0
#    NANDs: 0
#    NORs: 0
#    NXORs: 0
#    DFF: 0
```

Use Case: Loading and Convert DIMACS File

1. File->Open->DIMACS

- DIMACS files are used to store undirected graphs and is a standard format to SAT solvers
- CNF extension implies Conjunctive Normal Form format



The basic input format is as follows. At the top you can have *comment lines* that start with a c, like this:

```
c This line is a comment.
```

Then comes the *problem line*, which starts with a p and then says how many variables and clauses there are. For instance, here is a problem line that says this is a CNF problem with 3 variables and 4 clauses:

```
p cnf 3 4
```

Finally the clauses are listed. Each clause is represented as a list of numbers like 3 and -42. A positive number like 3 represents a positive occurrence of variable 3. A negative number like -42 represents a negated occurrence of variable 42.

The number 0 is treated in a special way: it is not a variable, but instead marks the end of each clause. This allows a single clause to be split up over multiple lines.

```
c  quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

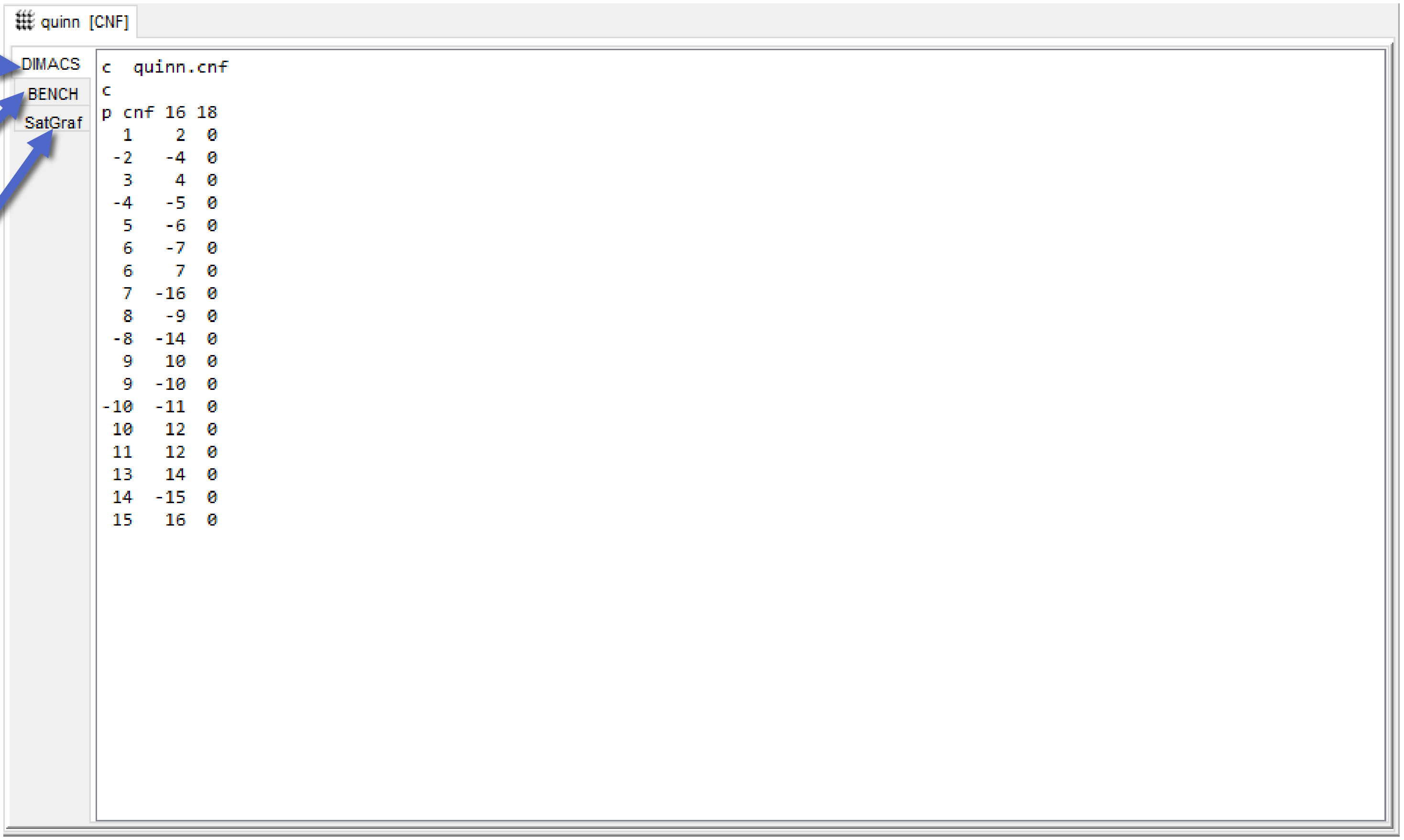
https://www.cs.utexas.edu/users/moore/acl2/manuals/current/manual/index-seo.php/SATLINK____DIMACS

Use Case: Loading and Convert DIMACS File

DIMACs file

Convert to BENCH

View SatGraf representation of CNF formula



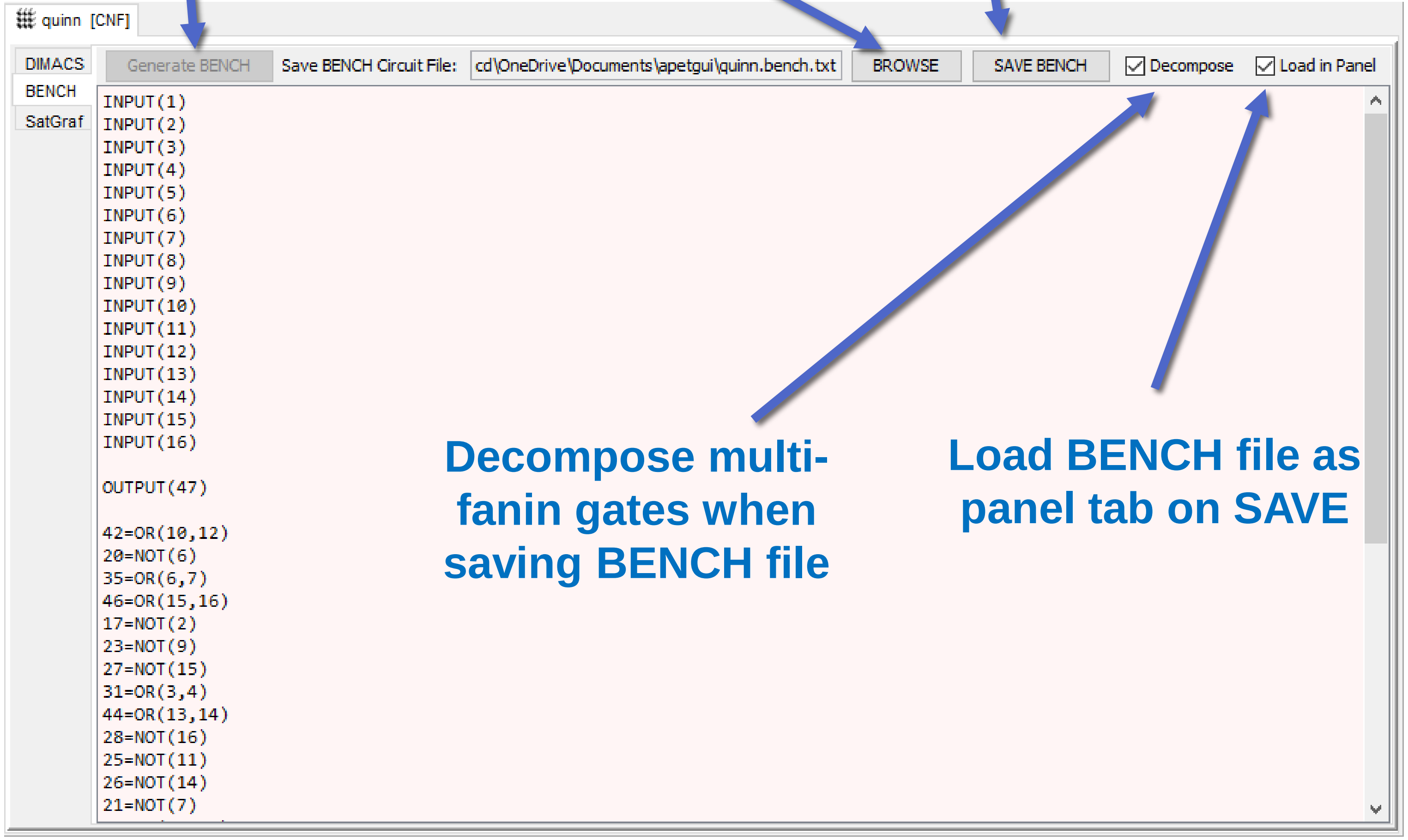
```
c  quinn.cnf
c
p cnf 16 18
  1    2  0
 -2   -4  0
  3    4  0
 -4   -5  0
  5   -6  0
  6   -7  0
  6    7  0
  7  -16  0
  8   -9  0
 -8  -14  0
  9   10  0
  9  -10  0
-10  -11  0
 10   12  0
 11   12  0
 13   14  0
 14  -15  0
 15   16  0
```

Use Case: Loading and Convert DIMACS File

1) Generate BENCH

2) BROWSE to choose filepath for BENCH text

3) SAVE BENCH to write BENCH text

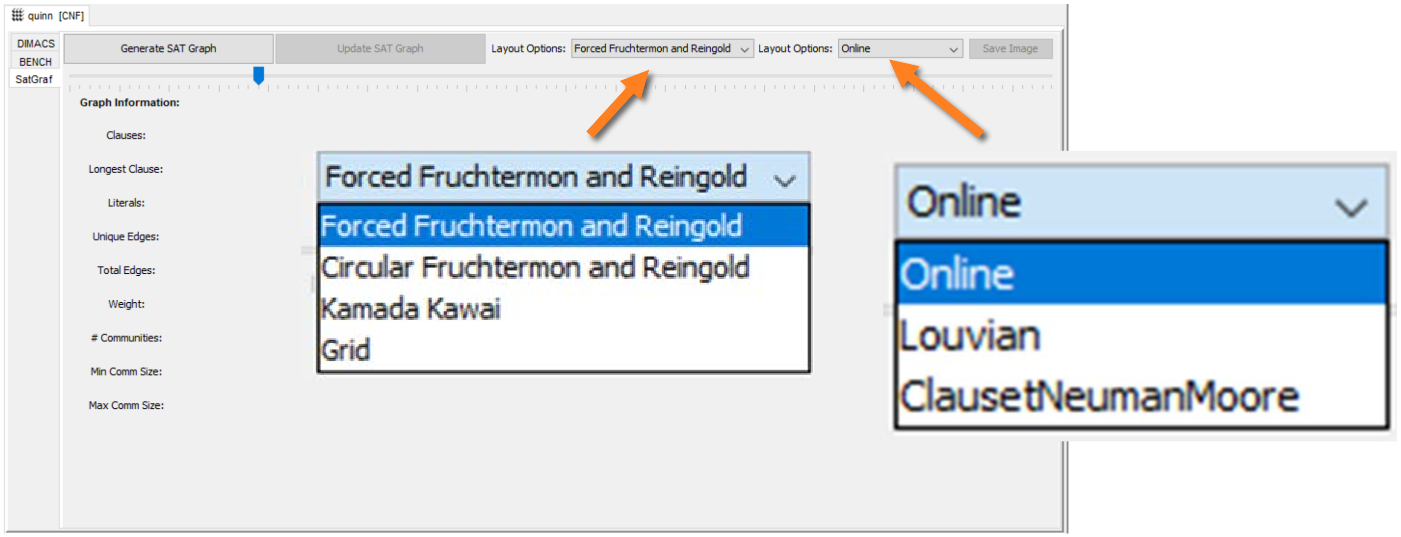


```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
INPUT(5)
INPUT(6)
INPUT(7)
INPUT(8)
INPUT(9)
INPUT(10)
INPUT(11)
INPUT(12)
INPUT(13)
INPUT(14)
INPUT(15)
INPUT(16)

OUTPUT(47)

42=OR(10,12)
20=NOT(6)
35=OR(6,7)
46=OR(15,16)
17=NOT(2)
23=NOT(9)
27=NOT(15)
31=OR(3,4)
44=OR(13,14)
28=NOT(16)
25=NOT(11)
26=NOT(14)
21=NOT(7)
```

Decompose multi-fanin gates when saving BENCH file

Load BENCH file as panel tab on SAVE

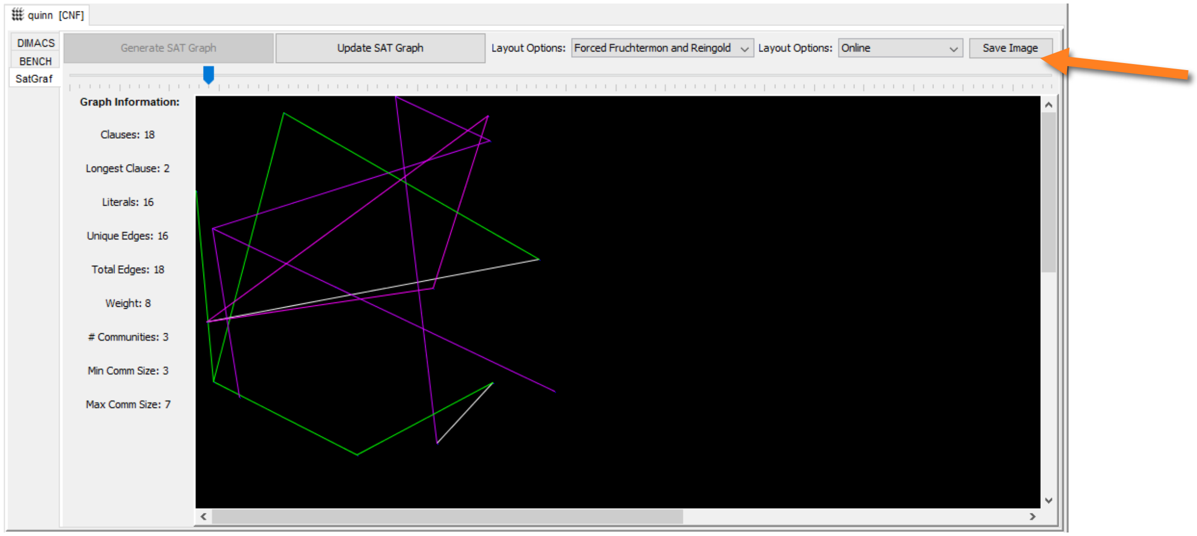
Use Case: Loading and Convert DIMACS File



Layout Options: Forced Fruchtermon and Reingold
- Forced Fruchtermon and Reingold
- Circular Fruchtermon and Reingold
- Kamada Kawai
- Grid

Layout Options: Online
- Online
- Louvian
- ClausetNeumanMoore

SatGraf community viewer: 4 layout options and 3 community formats



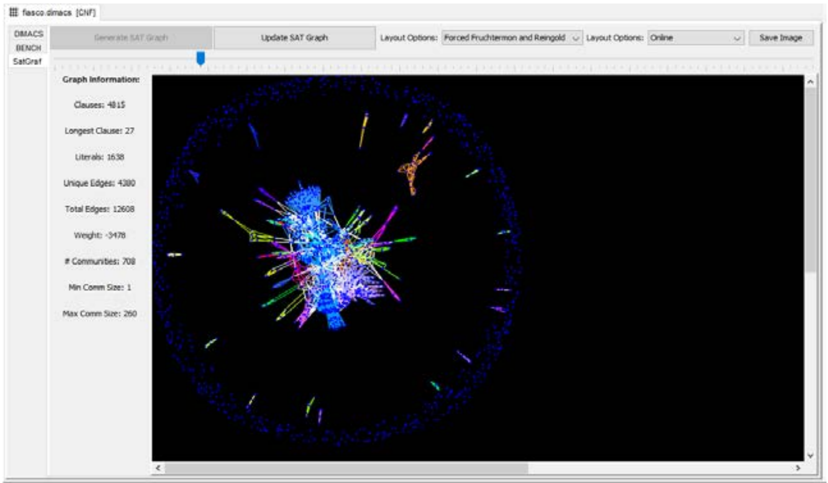
Layout Options: Forced Fruchtermon and Reingold

Layout Options: Online

Graph Information:
- Clauses: 18
- Longest Clause: 2
- Literals: 16
- Unique Edges: 16
- Total Edges: 18
- Weight: 8
- # Communities: 3
- Min Comm Size: 3
- Max Comm Size: 7

Save image of SatGraf

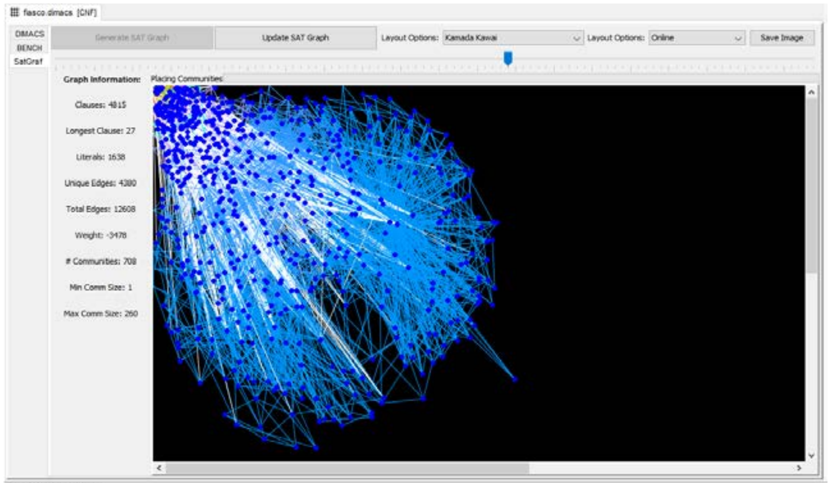
Use Case: Loading and Convert DIMACS File


fiasco.dimacs [CNF]

Layout Options: Forced Fruchtermon and Reingold
Layout Options: Online

Graph Information:

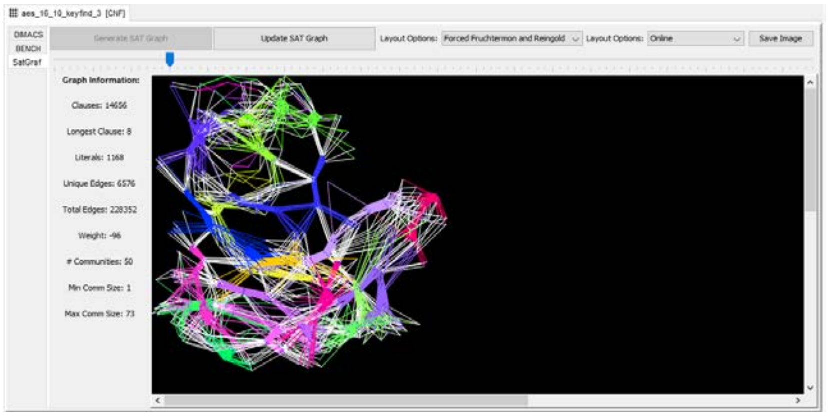
Clauses: 4815
Longest Clause: 27
Literals: 1638
Unique Edges: 4380
Total Edges: 12608
Weight: -3478
Communities: 708
Min Comm Size: 1
Max Comm Size: 260


fiasco.dimacs [CNF]

Layout Options: Kamada Kawai
Layout Options: Online

Graph Information: Placing Communities

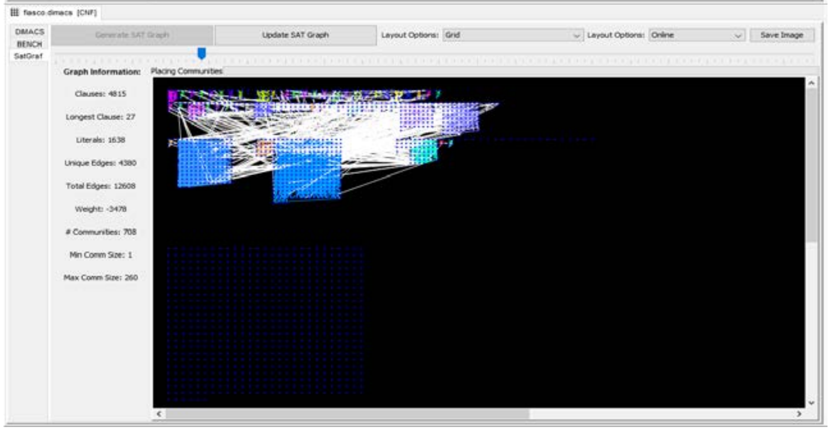
Clauses: 4815
Longest Clause: 27
Literals: 1638
Unique Edges: 4380
Total Edges: 12608
Weight: -3478
Communities: 708
Min Comm Size: 1
Max Comm Size: 260


aes_16_10_keyfind_3 [CNF]

Layout Options: Forced Fruchtermon and Reingold
Layout Options: Online

Graph Information:

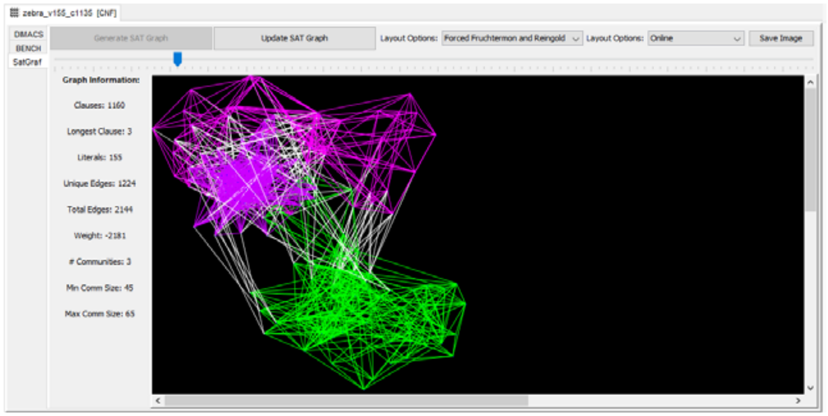
Clauses: 14656
Longest Clause: 8
Literals: 1168
Unique Edges: 6576
Total Edges: 228352
Weight: -96
Communities: 50
Min Comm Size: 1
Max Comm Size: 73


fiasco.dimacs [CNF]

Layout Options: Grid
Layout Options: Online

Graph Information: Placing Communities

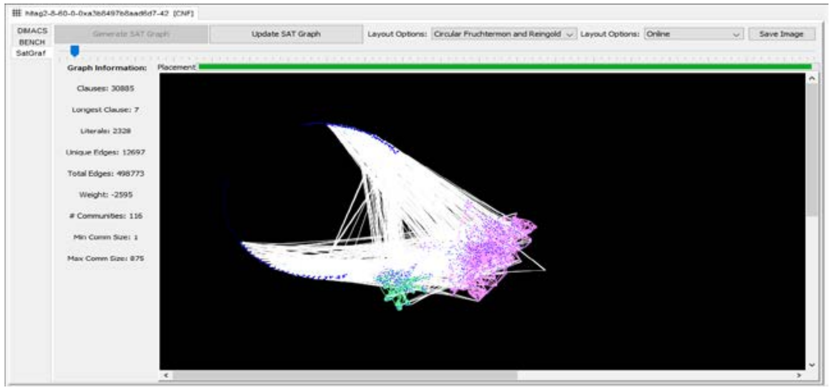
Clauses: 4815
Longest Clause: 27
Literals: 1638
Unique Edges: 4380
Total Edges: 12608
Weight: -3478
Communities: 708
Min Comm Size: 1
Max Comm Size: 260


zebra_v155_c1135 [CNF]

Layout Options: Forced Fruchtermon and Reingold
Layout Options: Online

Graph Information:

Clauses: 1160
Longest Clause: 3
Literals: 155
Unique Edges: 1224
Total Edges: 2144
Weight: -2181
Communities: 3
Min Comm Size: 45
Max Comm Size: 65


hitag2-8-60-0-0xa3b8497b8aad6d7-42 [CNF]

Layout Options: Circular Fruchtermon and Reingold
Layout Options: Online

Graph Information: Placement

Clauses: 30885
Longest Clause: 7
Literals: 2328
Unique Edges: 12697
Total Edges: 498773
Weight: -2595
Communities: 116
Min Comm Size: 1
Max Comm Size: 875

Tool Interfaces

Tool Interfaces

- **Espresso: version #2.3, Release date 01/31/88**
  - Computer program uses heuristic and specific algorithms for efficiently reducing complexity of digital electronic gate circuits
  - Copyright 1988 - 1983 by the Regents of the University of California
  - Part of the Octtools package for IC design developed at University of California, Berkeley
  - Richard Rudell published variant Espresso-MV in 1986 under paper *Multiple-Valued Logic Minimization for PLA Synthesis*.
  - PET uses ESPRESSO in native Windows format for logic and PLA minimization

  For more information see:
  https://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm

- **misII / MIS: release #2.2(AC)**
  - Algorithmic multi-level logic synthesis and minimization program
  - Starts from combinational logic macro-cell and produces optimized set of logic equations which preserves input-output behavior of the macro-cel
  - Has algorithms for minimizing area required to implement the logic equations
  - Has technology mapping step to map a network into a user specified cell library
  - Part of the Octtools package for IC design developed at the University of California, Berkeley
  - Copyright 1988 - 1983 by the Regents of the University of California
  - PET uses misII in native Windows format for gate synthesis in several algorithms.

  For more information see:
  https://embedded.eecs.berkeley.edu/pubs/downloads/octtools/index.htm

Tool Interfaces

- **ABC: version 1.01 (compiled Feb 13 2011 19:06:26)**
  - Software system for synthesis and verification of binary sequential logic circuits appearing in synchronous hardware designs
  - Combines scalable logic optimization based on And-Inverter Graphs (AIGs), optimal-delay DAG-based technology mapping for look-up tables and standard cells, and innovative algorithms for sequential synthesis and verification
  - Copyright (c) The Regents of the University of California. All rights reserved.
  - PET uses ABC for synthesis and processing of PLA and BLIF files as well as logic minimization and synthesis. PET also provides a graphical console interface for executing ABC scripts.

  For more information see:
  http://people.eecs.berkeley.edu/~alanmi/abc/

- **JDD: build 104, Feburary 2012**
  - A pure Java BDD and Z-BDD library - java implementation of decision diagram library inspired by BuDDy (BDD package written in C)
  - Includes support for Zero-suppressed BDD
  - Written by Arash Vahidi who provides software for use in academic projects
  - PET uses a modified version of the JDD library build 104, Feburary 2012, for generation and visualization of BDDs. Binary Decision Diagrams (BDDs) are used in formal verification, CSP and optimization..

  For more information see:
  https://bitbucket.org/vahidi/jdd/wiki/Home

Tool Interfaces

- **Z3 (version)**
  - The Satisfiability Modulo Theories (SMT) Solver Z3 supports the SMTLIB format. It is a theorem prover from Microsoft Research.
  - Licensed under the MIT license.
  - PET uses the native z3 Java library and Windows DLL for deriving models of single-output Boolean function circuits.

For more information see:

https://github.com/Z3Prover/z3/wiki

If you would like to see how z3 was used to solve a hardware-based CTF challenge see:

https://liveoverflow.com/minetest/

Tool Interfaces

- **SATGraf (version 0.2)**
  - Allows visualization of Boolean SAT instances in **DIMACS** format. It's primary purpose was to view the evolution of the structure of a Boolean SAT formula in real time as it is being processed by a conflict-driven clause-learning (CDCL) solver.
  - The tool is parametric, allowing the user to define the structure to be visualized. In particular, the tool can visualize the community structure of real-world Boolean satisfiability (SAT) instances and their evolution during solving.
  - Such visualizations have been the inspiration for several hypotheses about the connection between community structure and the running time of CDCL SAT solvers, some which we have already empirically verified.

  - For more information see:

  https://www.swmath.org/software/14761

  SATGraf was integrated partially into PET using the open source location at:

  https://bitbucket.org/znewsham/satgraf/src/master/

Tool Interfaces

- **Sat4j (version)**
  - Sat4j is a java library for solving Boolean satisfaction and optimization problems. It can solve SAT, MAXSAT, Pseudo-Boolean, Minimally Unsatisfiable Subset (MUS) problems.
  - Being in Java, the promise is not to be the fastest one to solve those problems (a SAT solver in Java is about 3.25 times slower than its counterpart in C++), but to be full featured, robust, user friendly, and to follow Java design guidelines and code conventions (checked using static analysis of the source code).
  - The library is designed for flexibility, by using heavily the decorator and strategy design patterns.
  - Sat4j is open source, under the dual business friendly Eclipse Public License and academic friendly GNU LGPL license.

  - For more information see:

  http://sat4j.org/

Native Graphics

- **yFiles for Java (version)**
  - PET utilizes the yWorks graph library, which is a Java-based toolkit for graph manipulation and visualization.
  - The primary LogicCircuit class in PET uses the Graph2D object as its core functionality for graph network operations.
  - PET can export graphics in native JPG format as well as **graphml**, which is the native format supported by the yEd graph editor program, made by yWorks.
  - You can download yEd viewer for graphml files at: https://www.yworks.com/