Logic Synthesis & Verification, Fall 2024

National Taiwan University

Programming Assignment 1

2 [Using ABC]

(a) *Create a BLIF file named "comp.blif" to represent a 5-to-3 compressor with 5 inputs x0, x1, x2, x3, and x4. Its output Y = (y2, y1, y0) is a 3- bit unsigned integer that represents the number of 1's in the inputs.*

We can establish the truth table by calculating the number of 1's for each possible input combination.

x4	x3	x2	x1	x0	y2	y1	y0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	0	1	0
0	1	1	0	1	0	1	1
0	1	1	1	0	0	1	1
0	1	1	1	1	1	0	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	1	1
1	0	1	0	0	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	0	0	1	1
1	0	1	1	1	1	0	0
1	1	0	0	0	0	1	0
1	1	0	0	1	0	1	1
1	1	0	1	0	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	0	0	1	1
1	1	1	0	1	1	0	0
1	1	1	1	0	1	0	0
1	1	1	1	1	1	0	1

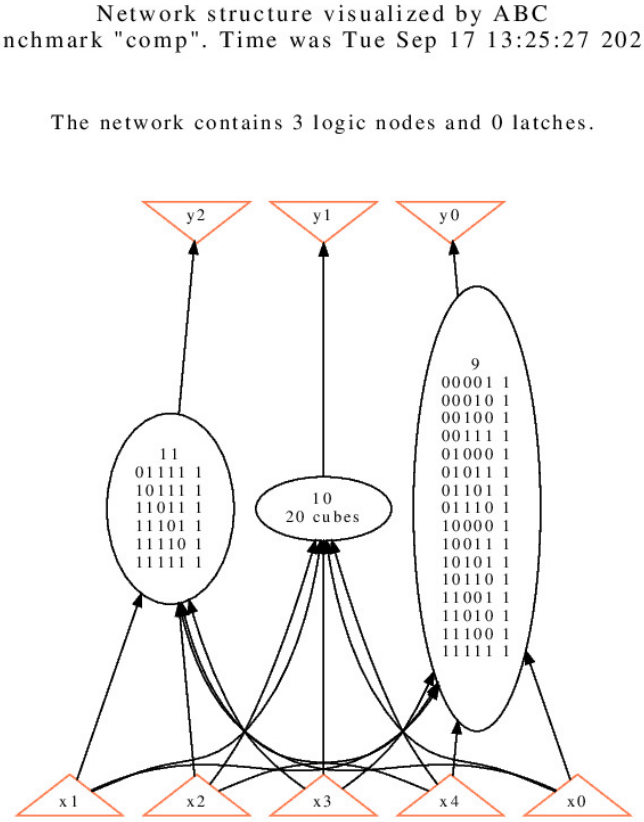
鄭小寶
Victor Zheng
A13921104

(b) *Perform the following steps to practice using ABC with your "comp.blif". Screenshot the results after running the commands and put them in your report.*

2. *check statistics (command "print stats")*

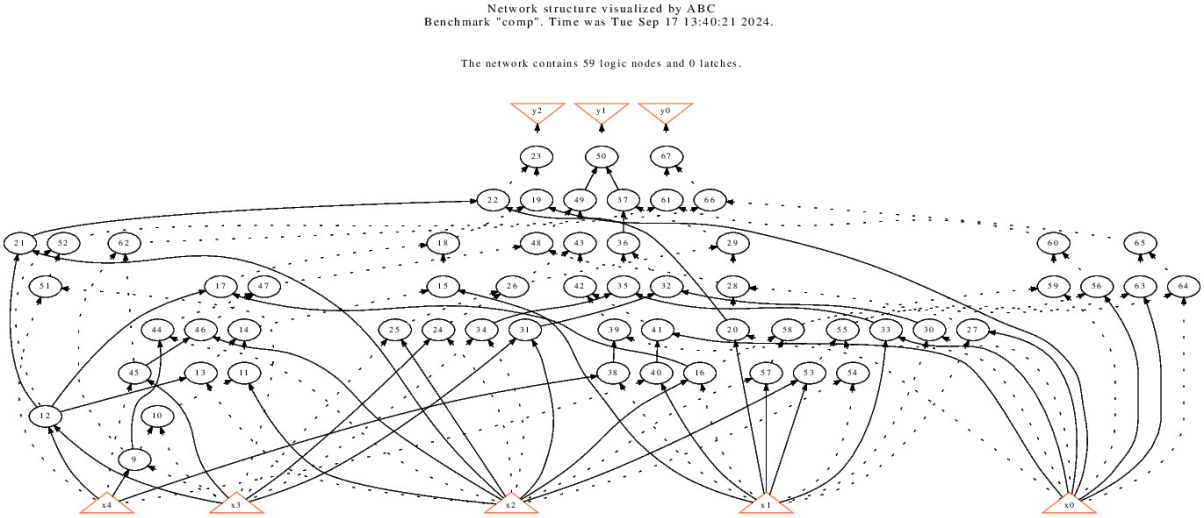
```
abc 01> read comp.blif
abc 02> print_stats
comp                        : i/o =    5/    3  lat =    0  nd =     3  edge =     15  cube =    42  lev = 1
```

3. *visualize the network structure (command "show")*

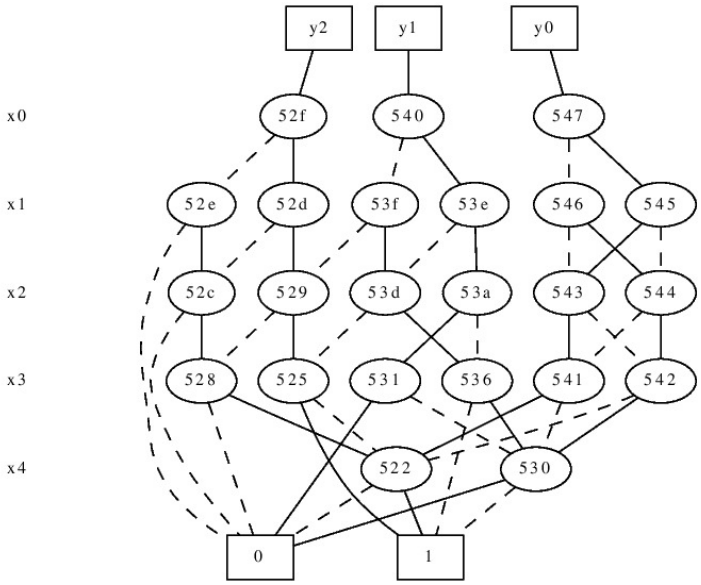


鄭小寶
Victor Zheng
A13921104

5. *visualize the AIG (command “show”)*



7. *visualize the BDD (command “show bdd -g”)*



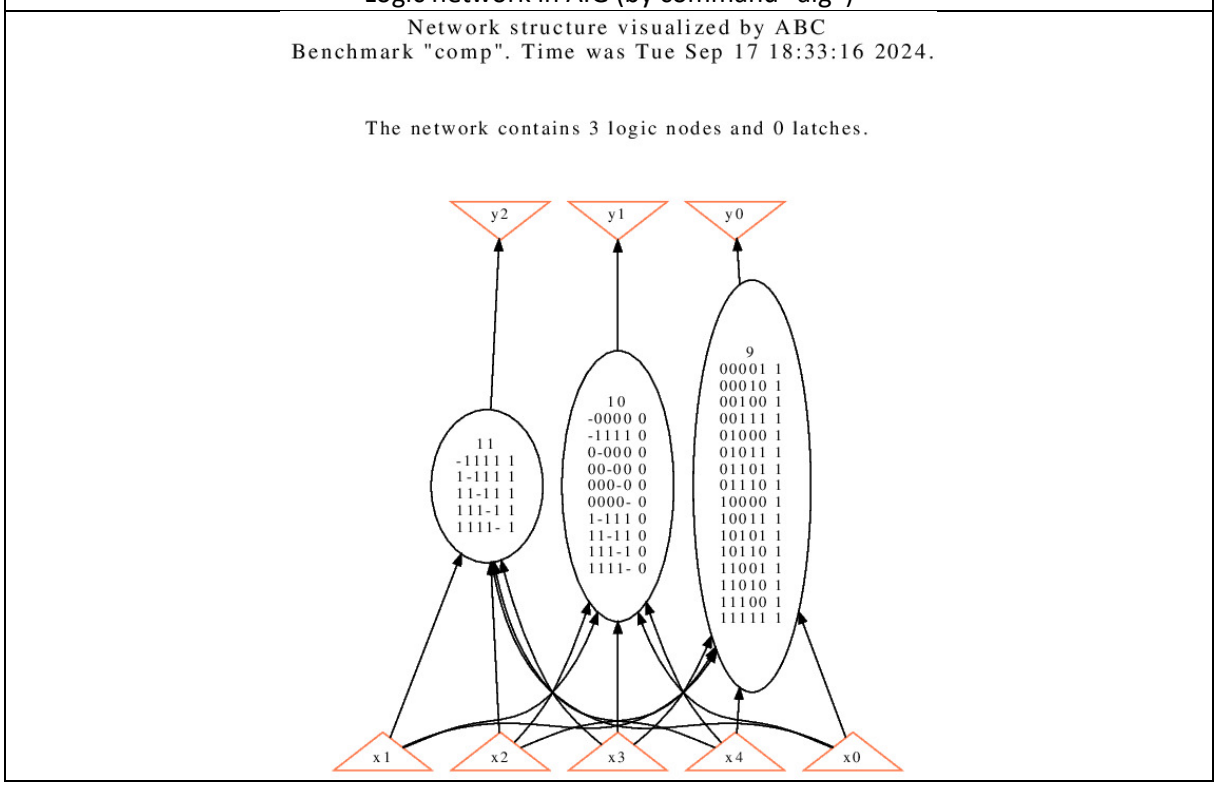
3 [ABC Boolean Function Representations]

(a) *Compare the following differences with your “comp.blif”. Screenshot the results and briefly describe your findings in your report.*

鄭小寶
Victor Zheng
A13921104

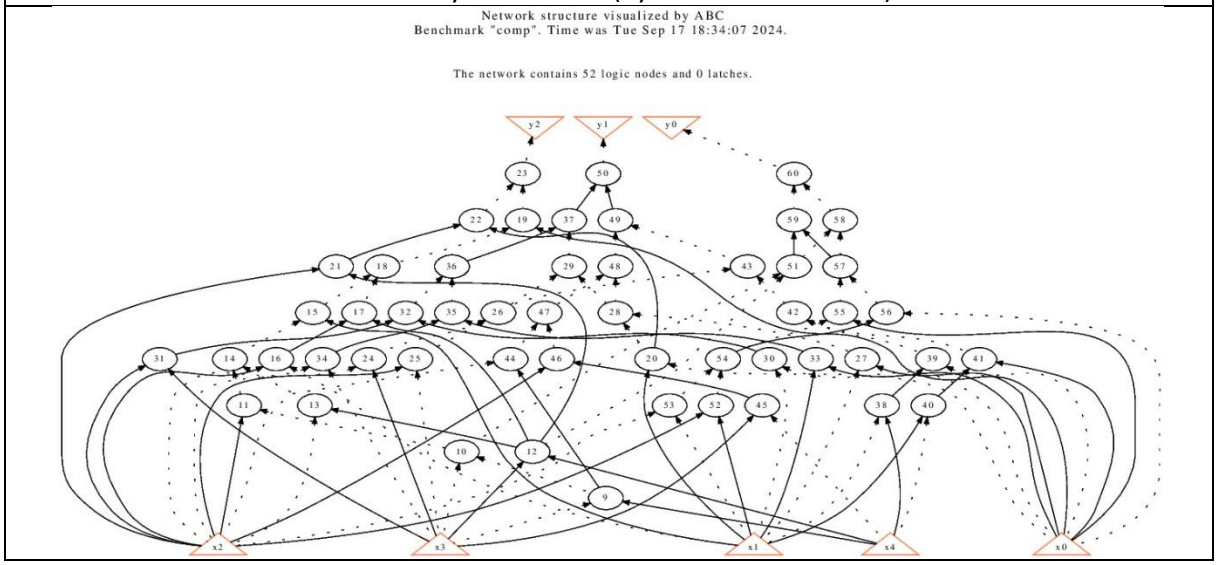
1. *logic network in AIG (by command “aig”) vs. structurally hashed AIG (by command “strash”)*

Logic network in AIG (by command “aig”)



The initial representation of the logic network as an And-Inverter Graph (AIG) is relatively compact, where nodes represent AND gates, and edges denote signals (possibly inverted).

Structurally hashed AIG (by command “strash”)



After applying the structural hashing process, instead of reducing the graph, the resulting structurally hashed AIG became significantly larger. This expansion is due to the reorganization and simplification of the internal logic, even though more nodes and edges were introduced.

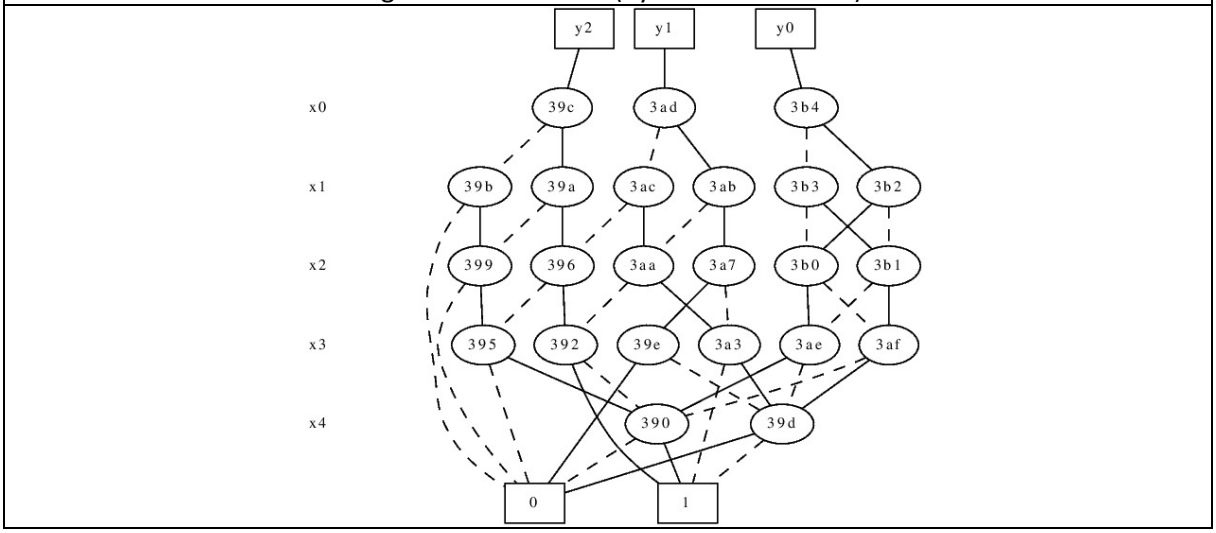
Surprisingly, after applying structural hashing, the graph became much larger. Despite the intention to simplify the network by merging equivalent subgraphs, the final hashed AIG expanded significantly. This suggests that, in this particular case, the hashing process led to a more complex graph structure,

鄭小寶
Victor Zheng
A13921104

potentially due to the reorganization of logic blocks or a different internal representation, though the logic itself was simplified.

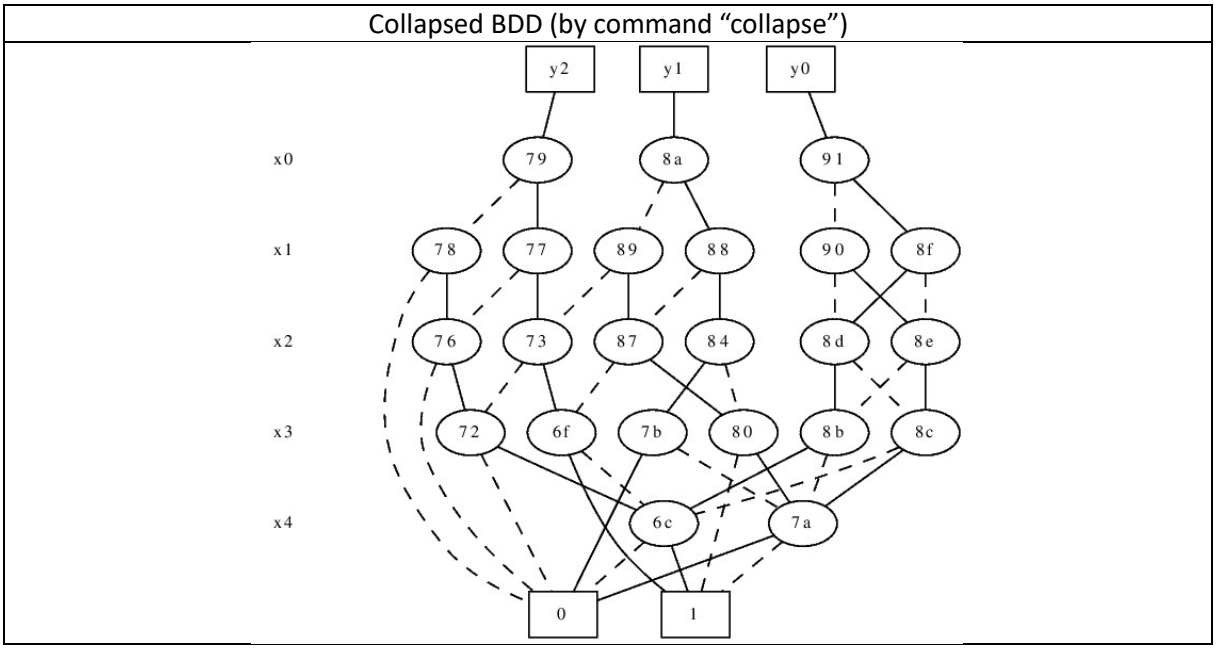
2. *logic network in BDD (by command “bdd”) vs. collapsed BDD (by command “collapse”)*

Logic network in BDD (by command “bdd”)



In Binary Decision Diagrams (BDD), the logic network is represented as a directed acyclic graph where each node represents a binary decision based on input variables.

Collapsed BDD (by command “collapse”)



After applying the collapse command, the overall structure of the BDD remained largely unchanged, with the same number of nodes. However, the values associated with these nodes were altered, indicating a reorganization of the internal logic without affecting the overall structure of the graph.

Upon collapsing the BDD, the number of nodes remained the same. The primary difference observed was that the node values were changed, reflecting a transformation in the decision process or logic representation. This suggests that the collapse operation did not simplify the network structurally but instead optimized the logic by reassigning values to existing nodes.

鄭小寶
Victor Zheng
A13921104

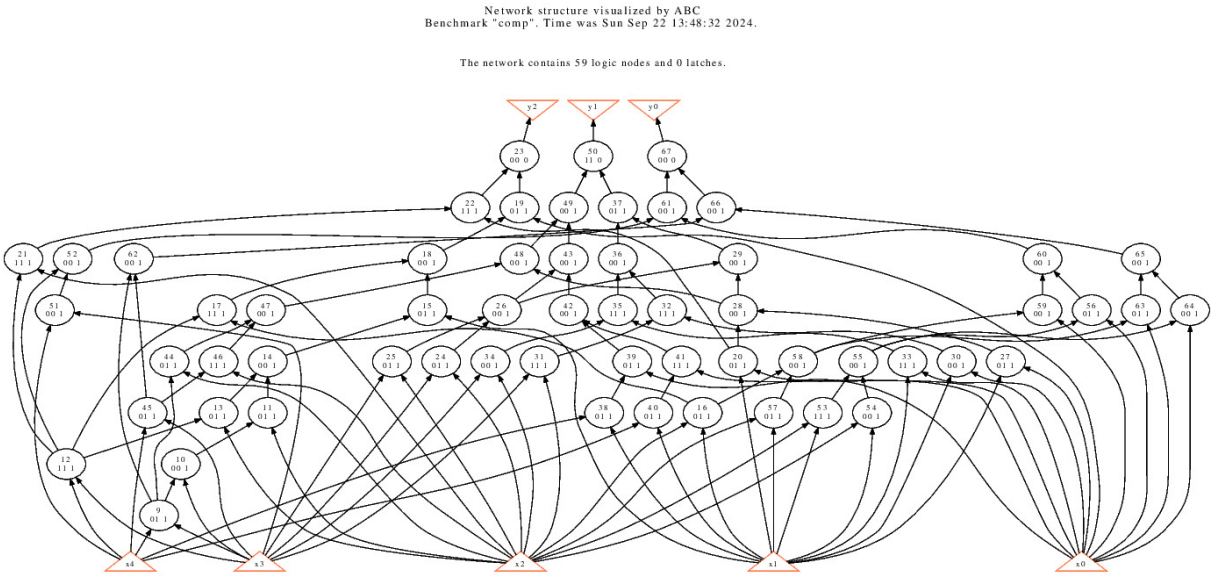
(b) *Given a structurally hashed AIG, find a sequence of ABC commands to convert it to a logic network with node function expressed in sum-of-products (SOP). Use your "comp.blif" to test your command sequence (by first running "strash" to convert it to AIG). Screenshot the results, and put them in your report.*

```
================================================
abc 01> read comp.blif
abc 02> ps
comp                        : i/o =    5/    3  lat =    0  nd =     3  edge =     15  cube =    42  lev = 1
abc 02> pf
    y0 = (((x3 # x4)((x0(x1 # x2)) + (!x0(x1 # x2)))) + ((x3 # x4)((x0(x1 # x2)) + (!x0(x1 # x2)))))
    y1 = (((x4 + !x4)(((x2 # x3)(x0 # x1)) + (((!x0!x1)(x2x3)) + ((x0x1)(!x2!x3))))) + (((x2 # x3)((!x0(!x1x4)) + (x0(x1!x4)))) + (((!x2(!x3x4)) + (x2(x3!x4)))(x0 # x1))))
    y2 = ((x0((x1((x2((x3(x4 + !x4)) + (!x3x4))) + (!x2(x3x4)))) + ((!x1x2)(x3x4)))) + ((!x0x1)(x2(x3x4))))
abc 02> strash
abc 03> ps
comp                        : i/o =    5/    3  lat =    0  and =    59  lev =  8
abc 03> pf
Error: Printing factored forms can be done for SOP networks.
abc 03> sop
Error: Converting to SOP is possible only for logic networks.
abc 03> logic
abc 04> sop
abc 04> ps
comp                        : i/o =    5/    3  lat =    0  nd =    59  edge =    118  cube =    59  lev = 8
```

```
abc 04> pf
    n9 = (!x3x4)
   n10 = (!x3!n9)
   n11 = (!n10x2)
   n12 = (x3x4)
   n13 = (!x2n12)
   n14 = (!n11!n13)
   n15 = (!n14x1)
   n16 = (!x1x2)
   n17 = (n12n16)
   n18 = (!n15!n17)
   n19 = (!n18x0)
   n20 = (!x0x1)
   n21 = (x2n12)
   n22 = (n20n21)
   n23 = (n19 + n22)
   n24 = (!x2x3)
   n25 = (!x3x2)
   n26 = (!n24!n25)
   n27 = (!x1x0)
   n28 = (!n20!n27)
   n29 = (!n26!n28)
   n30 = (!x0!x1)
   n31 = (x2x3)
   n32 = (n30n31)
   n33 = (x0x1)
   n34 = (!x2!x3)
   n35 = (n33n34)
   n36 = (!n32!n35)
   n37 = (!n29n36)
   n38 = (!x1x4)
   n39 = (!x0n38)
   n40 = (!x4x1)
   n41 = (x0n40)
   n42 = (!n39!n41)
   n43 = (!n26!n42)
   n44 = (!x2n9)
   n45 = (!x4x3)
   n46 = (x2n45)
   n47 = (!n44!n46)
   n48 = (!n28!n47)
   n49 = (!n43!n48)
   n50 = (!n37 + !n49)
   n51 = (!x3!x4)
   n52 = (!n12!n51)
   n53 = (x1x2)
   n54 = (!x1!x2)
   n53 = (x1x2)
   n54 = (!x1!x2)
   n55 = (!n53!n54)
   n56 = (!n55x0)
   n57 = (!x2x1)
   n58 = (!n16!n57)
   n59 = (!x0!n58)
   n60 = (!n56!n59)
   n61 = (!n52!n60)
   n62 = (!n9!n45)
   n63 = (!n58x0)
   n64 = (!x0!n55)
   n65 = (!n63!n64)
   n66 = (!n62!n65)
   n67 = (n61 + n66)
```


鄭小寶
Victor Zheng
A13921104

To convert a structurally hashed AIG into a logic network expressed in sum-of-products (SOP), I first loaded my "comp.blif" file into ABC and executed the strash command to transform the file into an AIG representation. Next, I used the logic command to optimize the logic network, followed by the sop command to express the node functions in terms of sum-of-products. This sequence of commands successfully generated a logical structure that accurately represents the desired functionality. The obtained results were captured in screenshots, which will be included in the report to illustrate the conversion process and the final outputs.



鄭小寶
Victor Zheng
A13921104