LSV Programming Assignment 1

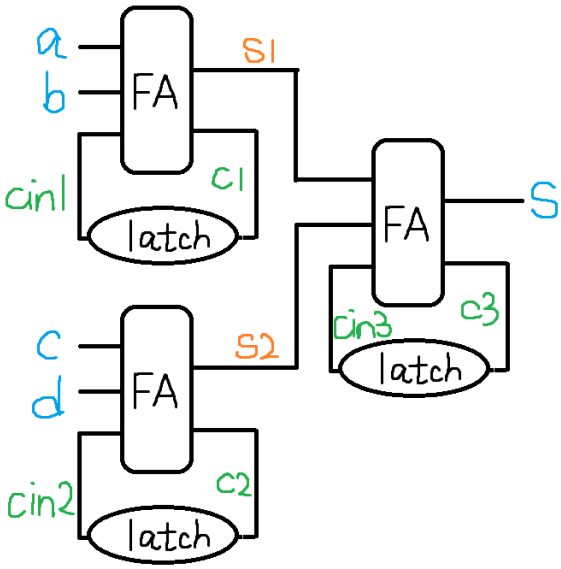
Student ID: R10943109 Name: 丁宣勻

PART I

(a) Brief view of my circuit design of a four-number serial adder and the BLIF file.

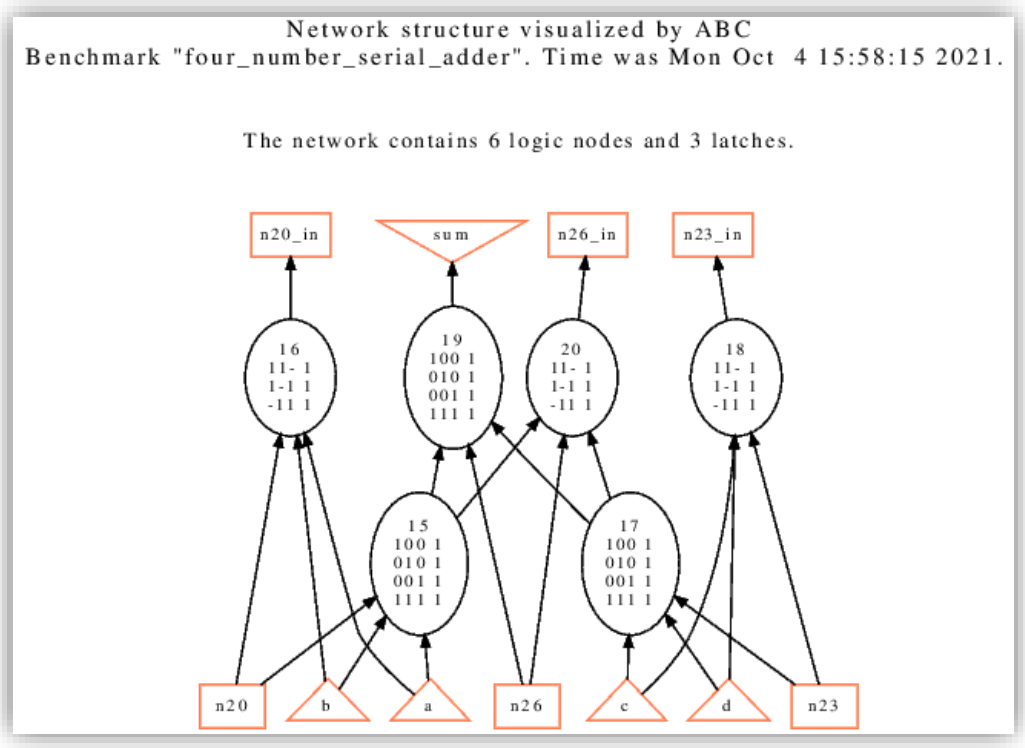
```
.model four_number_serial_adder
.inputs a b c d
.outputs sum
#
.subckt full_adder a=a  b=b  cin=cin1 sum=s1  cout=c1
.subckt full_adder a=c  b=d  cin=cin2 sum=s2  cout=c2
.subckt full_adder a=s1 b=s2 cin=cin3 sum=sum cout=c3
#
.latch c1 cin1 0
.latch c2 cin2 0
.latch c3 cin3 0
.end

.model full_adder
.inputs a b cin
.outputs sum cout
# s = XOR(a, b, cin)
.names a b cin sum
100 1
010 1
001 1
111 1
# cout = MAJ(a, b, cin)
.names a b cin cout
11- 1
1-1 1
-11 1
.end
```

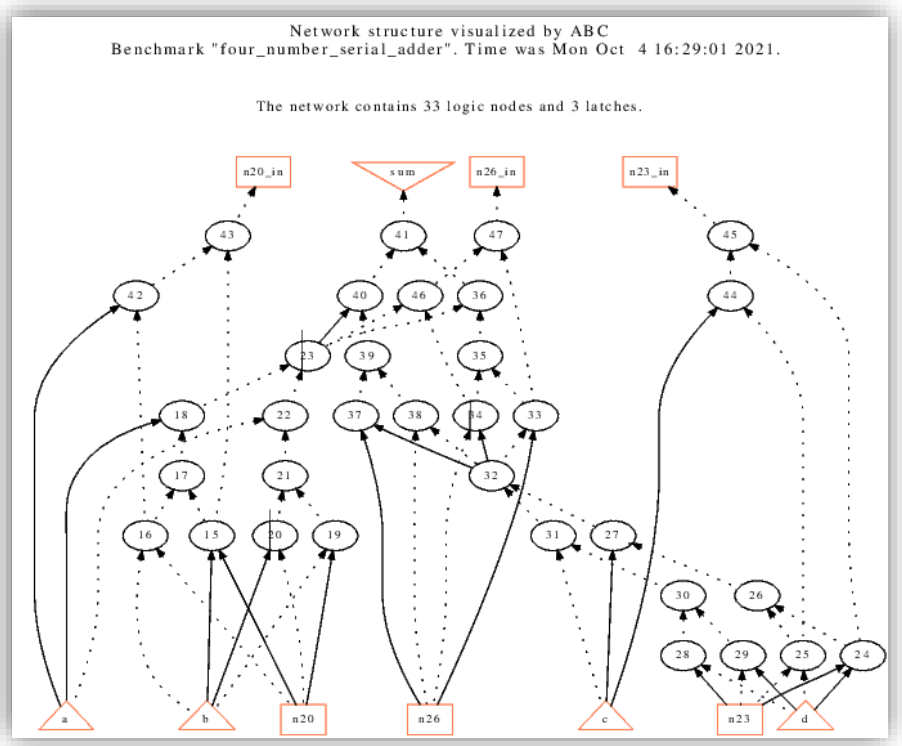


(b) Step 3, 5, 7

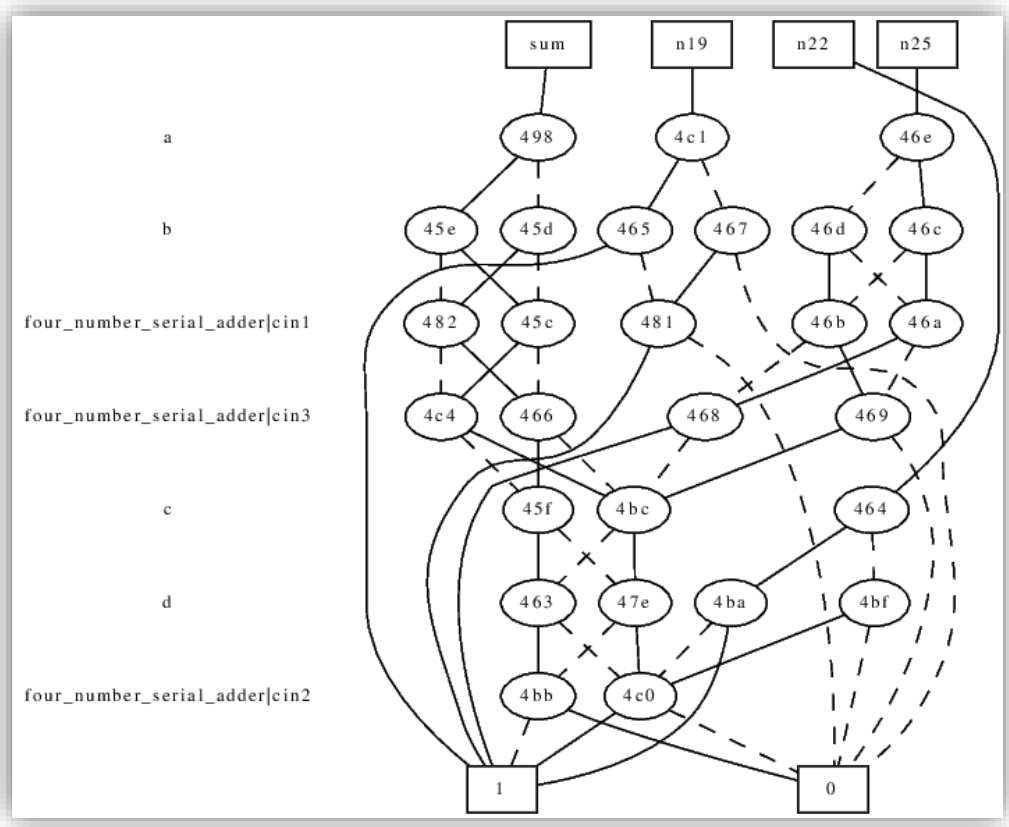
- Step 3: visualize the network structure (command show).



- Step 5: visualize the AIG (command show).



- Step 7: visualize the BDD (command show_bdd -g; note that show_bdd only shows the first PO; option -g can be applied to show all POs)



PART II

(a) Compare the following differences with the four-number serial adder example.

1. logic network in AIG (by command "aig") vs. structurally hashed AIG (by command "strash")

According to the website(http://people.eecs.berkeley.edu/~alanmi/abc/),

- aig – Converts local functions of the nodes to AIG.
- strash – Transforms the current network into an AIG by one-level structural hashing. The resulting AIG is a logic network composed of two-input AND gates and inverters represented as complemented attributes on the edges. Structural hashing is a purely combinational transformation, which does not modify the number and positions of latches.

In addition to aig and strash commands, there is another command called renode,

- renode – Assumes that the input is an AIG. Creates node boundaries in this AIG and collapses the intermediate logic to form larger nodes.

Take four-number serial adder for an example:

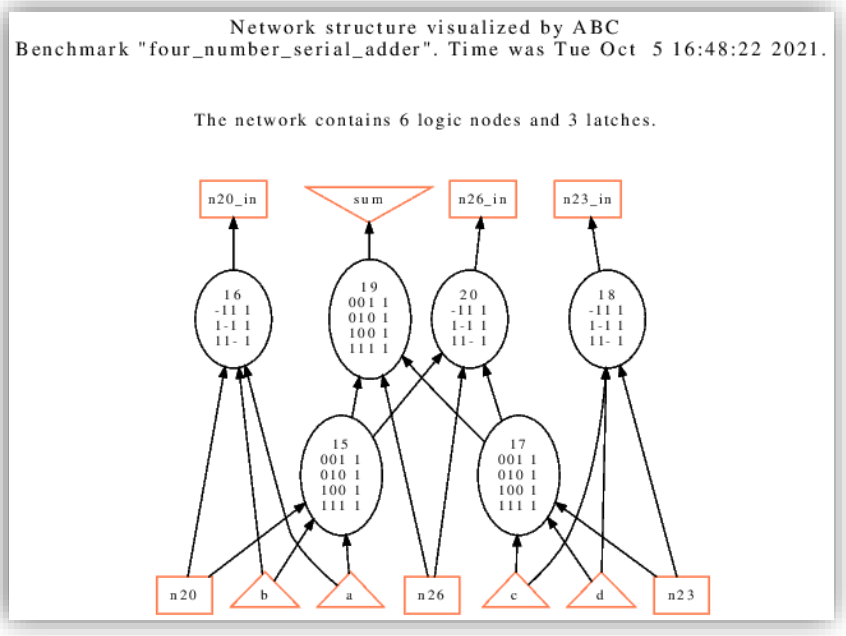
After reading the BLIF file, print_stats shows it has 21 cubes in my four-number serial adder.

```
abc 08> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  nd =     6  edge =     18  cube =    21  lev = 2
```

After aig converts local functions of the nodes to AIGs, print_stats shows there are 39 aig instances in my four-number serial adder.

```
abc 08> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  nd =     6  edge =     18  aig  =    39  lev = 2
```

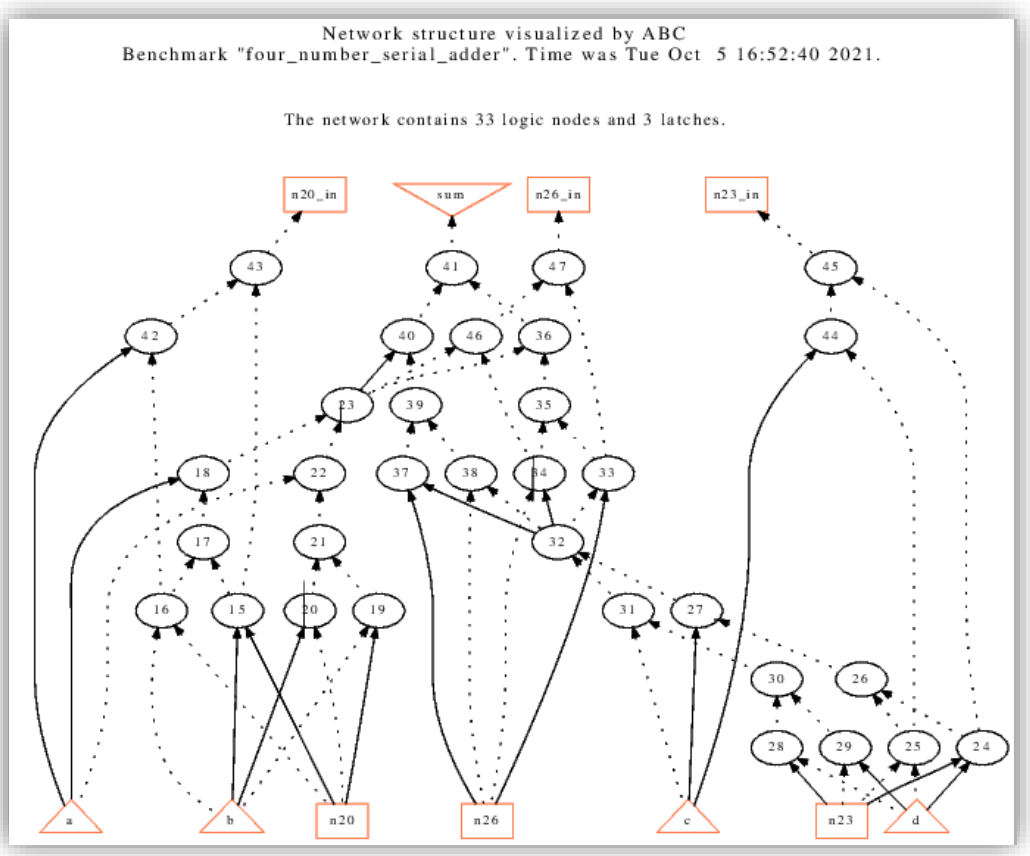
However, using show after aig command can only show the circuit with SOP logic network.



strash command can be used to convert the initial logic network into an AIG using structural hashing. print_stats shows that 6 function nodes in the previous pictures are replaced by 33 AND gates, and there are no edge stats in strash form.

```
abc 10> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  and =     33  lev =  8
```

The AIG of the whole network can be showed by show command after applying strash command.



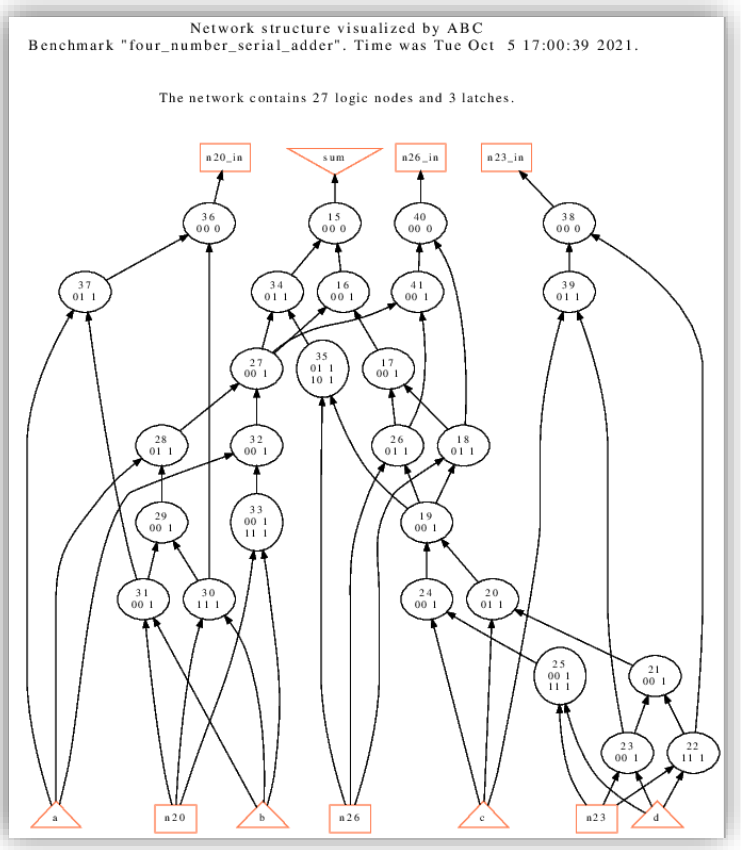
In short, aig only converts “node” functions to AIG; on the other hand, strash converts the “whole” network to AIG form. Therefore, AIG (And-Inverter Graph) of a network can only be constructed and shown after strash command.

Notice there is another command called renode, which can recreate SOP logic network from the AIG. However, the recreated SOP logic network will be different from the original one, which was initialized by the BLIF file.

print_stats after renode:

```
abc 11> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  nd =    27  edge =     54  aig  =    33  lev = 8
```

show after renode:



2. logic network in BDD (by command "bdd") vs. collapsed BDD (by command "collapse")

According to the website(http://people.eecs.berkeley.edu/~alanmi/abc/),

- bdd – Converts local functions of the nodes to BDDs.
- collapse – Recursively composes the fanin nodes into the fanout nodes resulting in a network, in which each CO is produced by a node, whose fanins are CIs. Collapsing is performed by building global functions using BDDs and is, therefore, limited to relatively small circuits. After collapsing, the node functions are represented using BDDs.

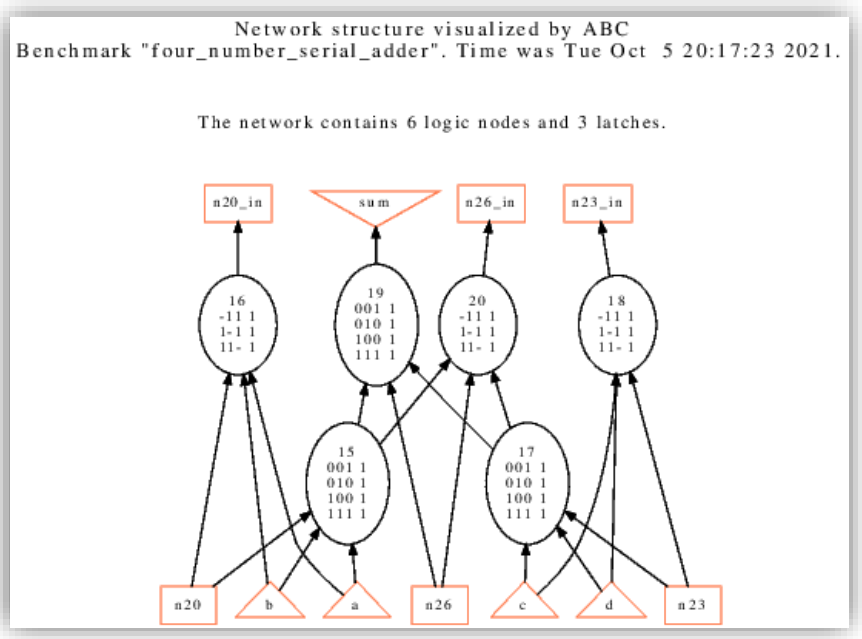
Take four-number serial adder for an example:
After reading the BLIF file, print_stats shows it has 21 cubes in my four-number serial adder.

```
abc 08> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  nd =     6  edge =    18  cube =    21  lev = 2
```

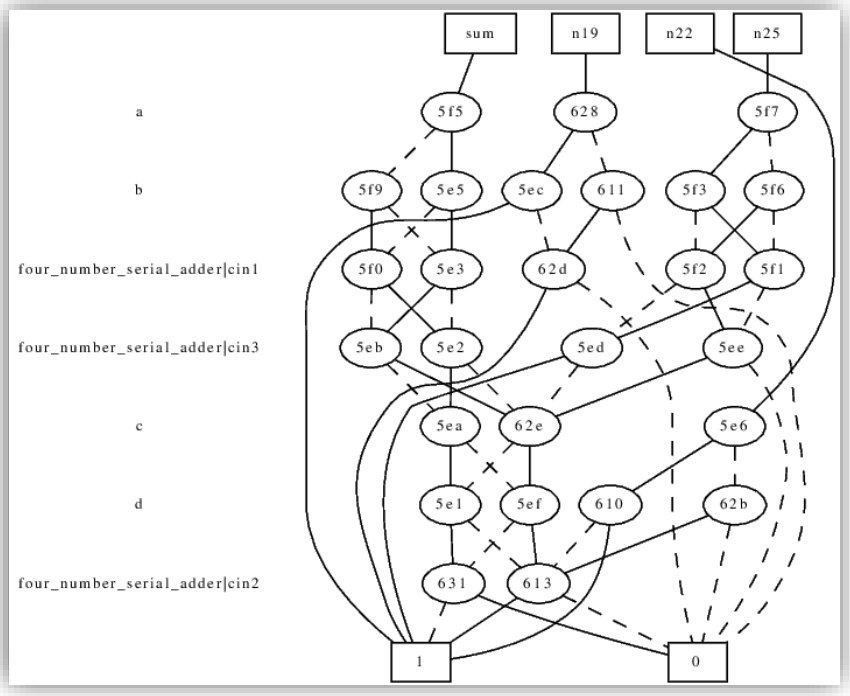
After bdd converts local functions of the nodes to BDDs, print_stats shows there are 21 bdd instances in my four-number serial adder.

```
abc 18> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  nd =     6  edge =    18  bdd  =    21  lev = 2
```

Use show after bdd command to visualize the network. The diagram is the same with the original one.



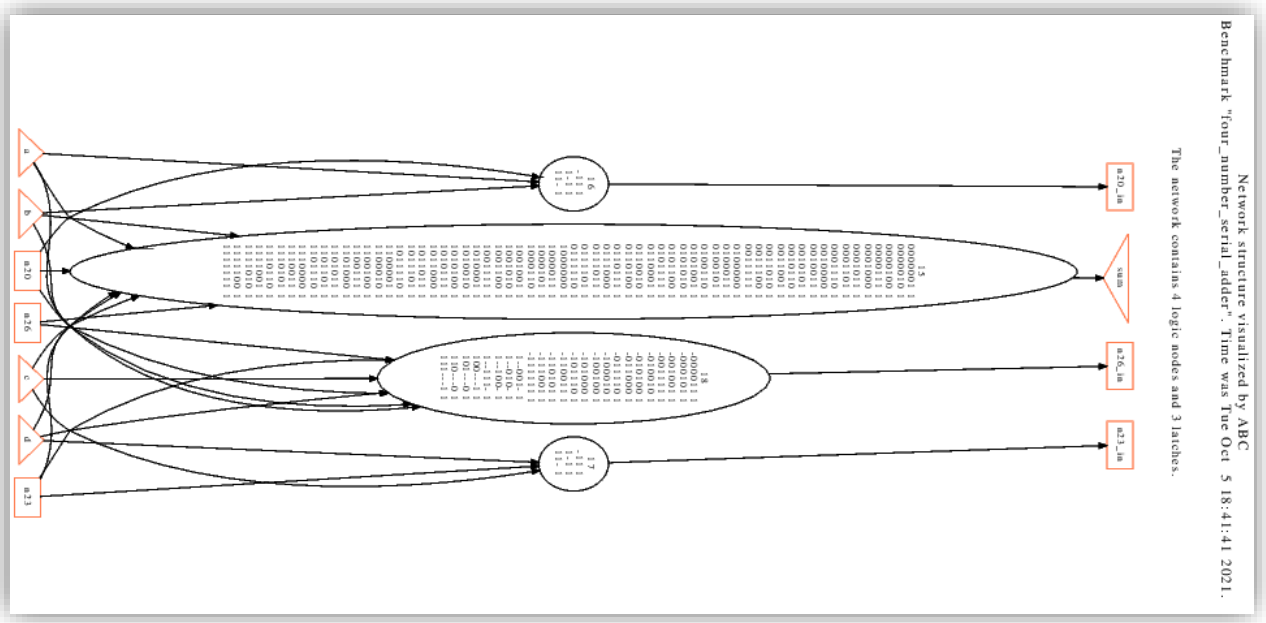
Use show_bdd -g after bdd command to visualize the BDD.



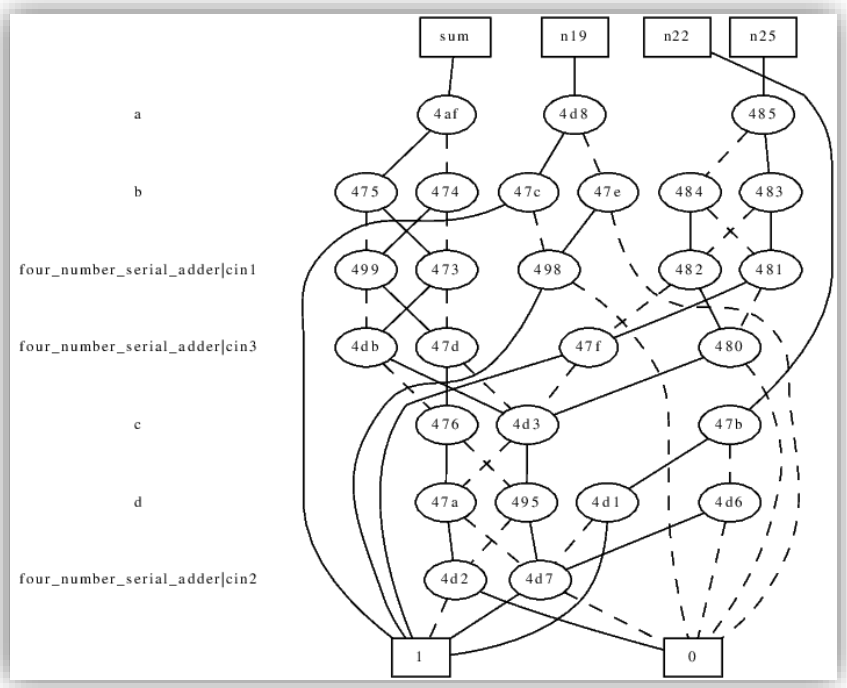
collapse collapses the network by constructing global BDDs. print_stats shows there are 25 bdd instances in my four-number serial adder after collapse.

```
abc 19> print_stats
four_number_serial_adder      : i/o =    4/    1  lat =    3  nd =     4  edge =     20  bdd  =    25  lev = 1
```

The network showed by show command after applying collapse command. Compared with the previous show command, the levels had collapsed into a single level as you can tell from the diagram.



show_bdd -g after collapse command is still the same, which is not surprised because the inputs/outputs relation is still the same.



In short, bdd only converts "node" functions to BDD; on the other hand, collapse will construct a global BDD tree by collapsing the "whole" network, and thus generate a one-level diagram.

(c) 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).

There are 2 commands that can convert AIG into SOP representations: logic and renode.
Here are the explanations from the website(http://people.eecs.berkeley.edu/~alanmi/abc/):

- logic – Transforms the AIG into a logic network with the SOP representation of the two-input AND-gates.
- renode – Assumes that the input is an AIG. Creates node boundaries in this AIG and collapses the intermediate logic to form larger nodes.

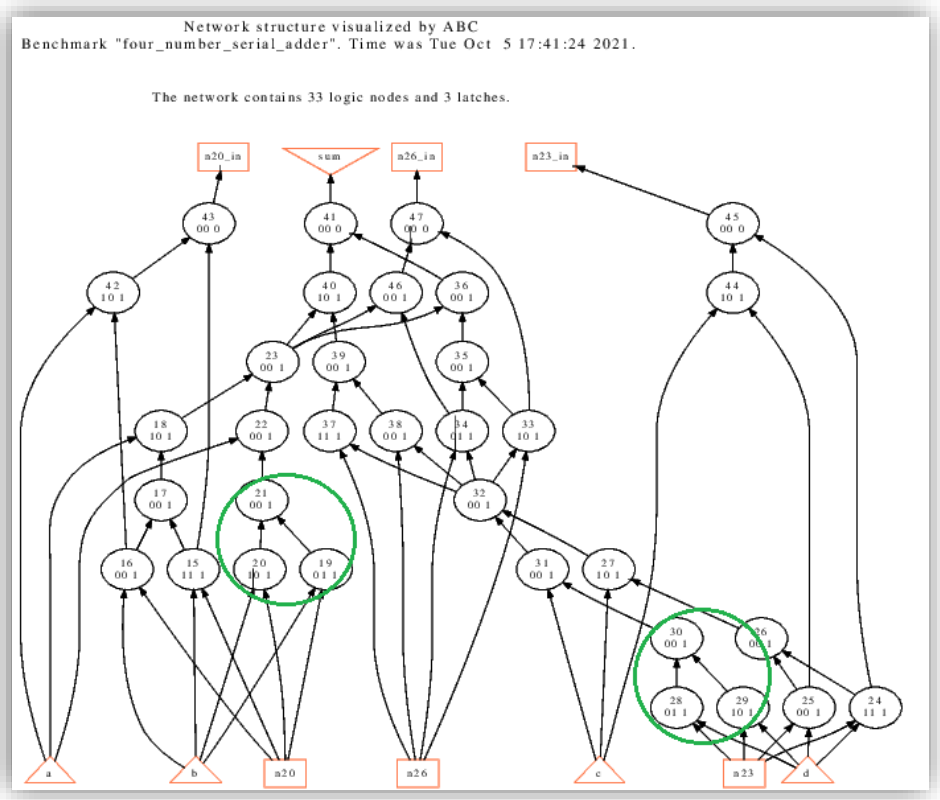
Here is the help information of 2 commands:

- logic – Transform an AIG into a logic network with SOPs.
- renode – Transforms the AIG into a logic network with larger nodes while minimizing the number of FF literals of the node SOPs.

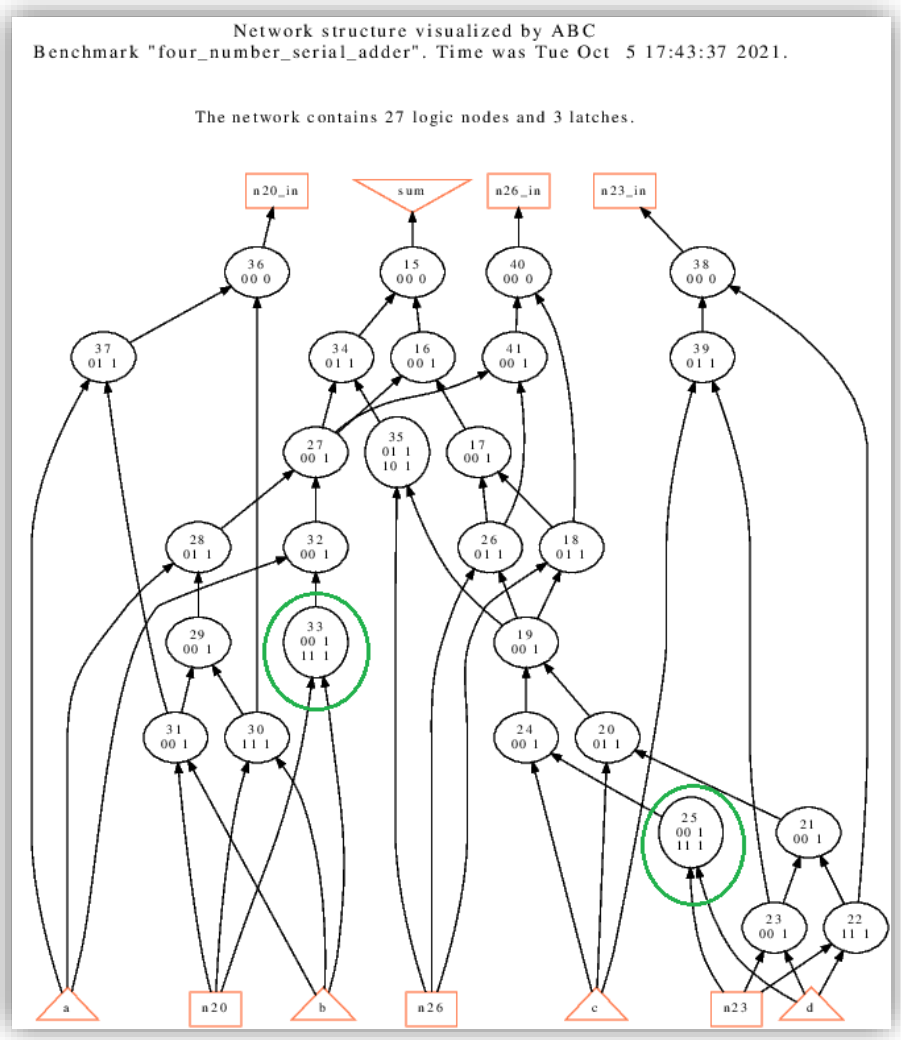
The main difference is that renode will minimize the number of FF literals by combining multiple nodes into a larger node.

Take my four-number serial adder for an example:

- logic



- renode



Both diagrams are in SOP form. You can see that the SOP constructed by renode command may generate bigger nodes (in green circles).

MY ABC NOTE

ABC: A System for Sequential Synthesis and Verification

- Primary Goal: to keep data structures simple and flexible for a wide range of applications.
- Basic Premises:
  1. Allow for a variety of functional representations, such as BDDs and SOPs, while defaulting to AIGs for the mainstream network manipulation.
  2. Synergy between synthesis and verification using efficient SAT-based Boolean reasoning on AIG for combinational and sequential equivalence checking.
  3. Public-domain implementation of the state-of-the-art combinational and sequential synthesis algorithms.
  4. Open-source environment.