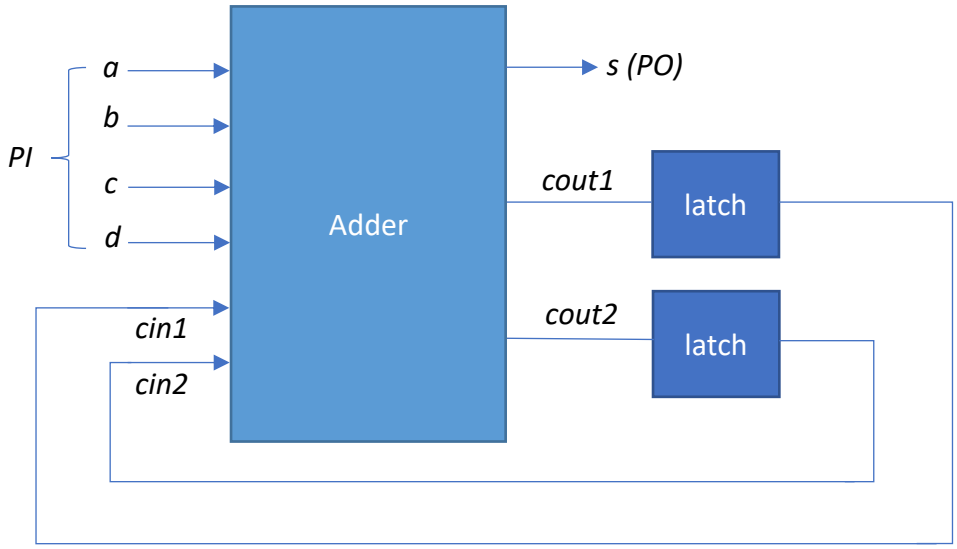
Logic Synthesis & Verification, Fall 2021 National Taiwan University

Programming Assignment 1 Report

R10943093 謝秉翰

1 [Using ABC]

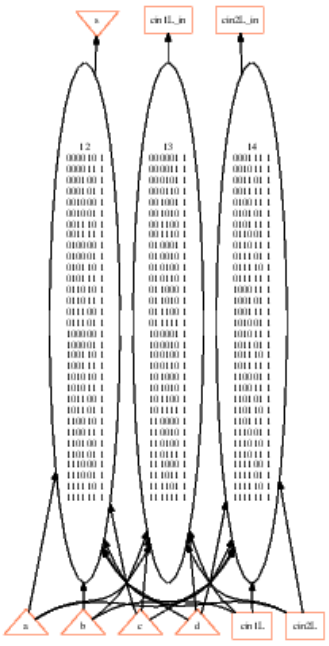
- **BLIF file:** *Four_number_SA.blif*
- **Architecture:** *Four-number serial adder:*



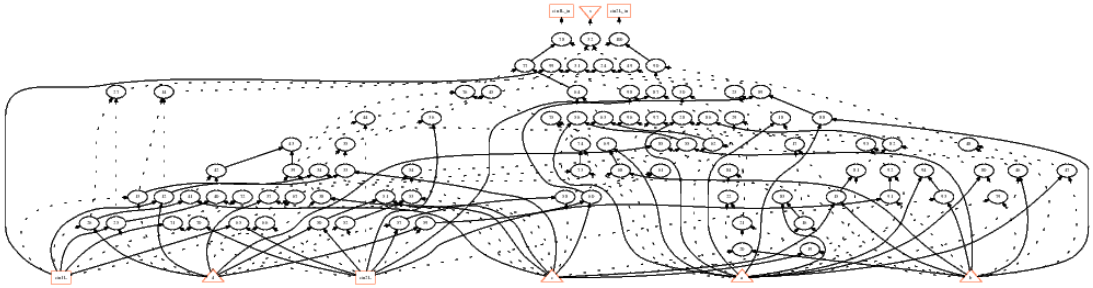
- **Truth table** is completed in appendix.
- **Commands and Results**:
  1. read
  2. print_stats

```
four_number_serial_adder      : i/o =    4/    1  lat =    2  nd =     3  edge =     18  cube =    96  lev = 1
```

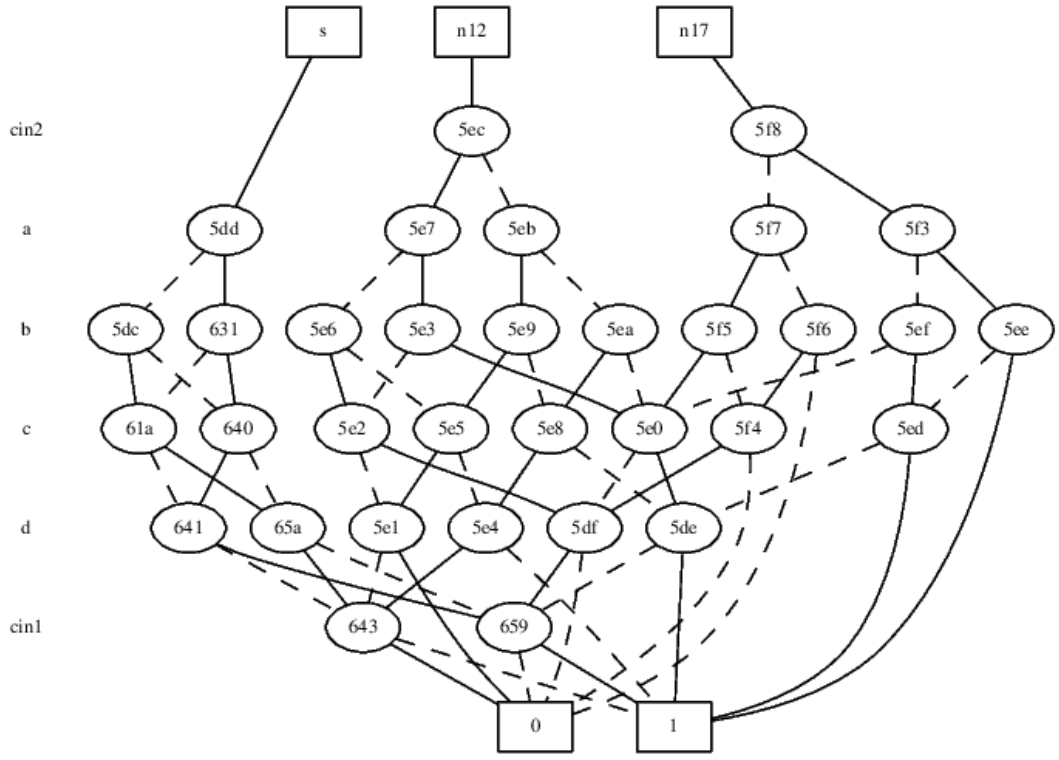
  3. show



4. strash
5. show



6. collapse
7. show_bdd -g



2 [ABC Boolean Function Representations]

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

1. **The difference between these two commands is that command "aig" only convert node functions to AIG, while command "strash" transforms the whole combinational logic into AIG.**

   Here are the explanations and helps for the commands,

   aig – Converts local functions of the nodes to AIGs.

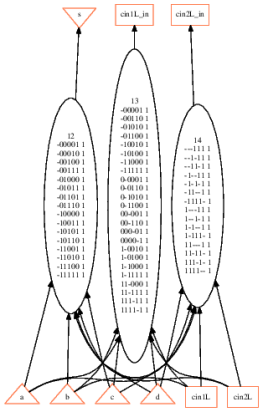
```
abc 07> aig -h
usage: aig [-h]
                converts node functions to AIG
        -h    : print the command usage
```

   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.

```
abc 07> strash -h
usage: strash [-acrih]
                transforms combinational logic into an AIG
        -a    : toggles between using all nodes and DFS nodes [default = DFS]
        -c    : toggles cleanup to remove the dagling AIG nodes [default = all]
        -r    : toggles using the record of AIG subgraphs [default = no]
        -i    : toggles complementing the POs of the AIG [default = no]
        -h    : print the command usage
```

   We can observe the results by the commands of "show", "print_stats".

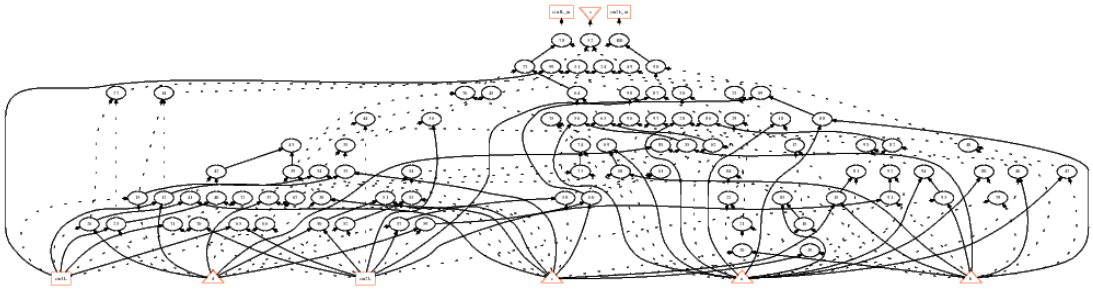
   By applying the command "aig", the "nd", "edge", and "lev" are still the same, the only different is that it has "aig = 110" (below), which is the AIG of the node function. You can also notice that the structure of the graph is still nearly the same (right).



```
four_number_serial_adder      : i/o =    4/    1  lat =    2  nd =     3  edge =     18  aig =   110  lev = 1
```

   However, by applying the command "strash", the combinational logic transforms to AIG, the graph is shown below and there are 89 AND with the # level equals to 9. (The # of latch will not change)

```
four_number_serial_adder      : i/o =    4/    1  lat =    2  and =     89  lev =  9
```



2. **The difference between these two commands is that command “bdd” only convert node functions to BDD, while command “collapse” collapse the network by constructing global BDDs.**

   Here are the explanations and helps for the commands,

   bdd – Converts local functions of the nodes to BDDs.

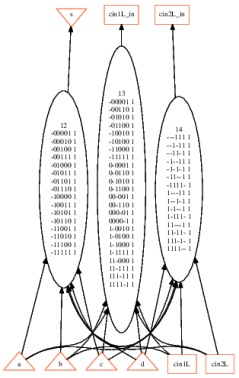
```
abc 07> bdd -h
usage: bdd [-rsh]
                converts node functions to BDD
        -r      : toggles enabling dynamic variable reordering [default = yes]
        -s      : toggles constructing BDDs directly from SOPs [default = no]
        -h      : print the command usage
```

   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.

```
abc 07> collapse -h
usage: collapse [-B <num>] [-L file] [-rodxvh]
                collapses the network by constructing global BDDs
        -B <num>: limit on live BDD nodes during collapsing [default = 1000000000]
        -L file : the log file name [default = no logging]
        -r      : toggles dynamic variable reordering [default = yes]
        -o      : toggles reverse variable ordering [default = no]
        -d      : toggles dual-rail collapsing mode [default = no]
        -x      : toggles dumping file "order.txt" with variable order [default = no]
        -v      : print verbose information [default = no]
        -h      : print the command usage
```

   We can observe the results by the commands of “show”, “print_stats”.

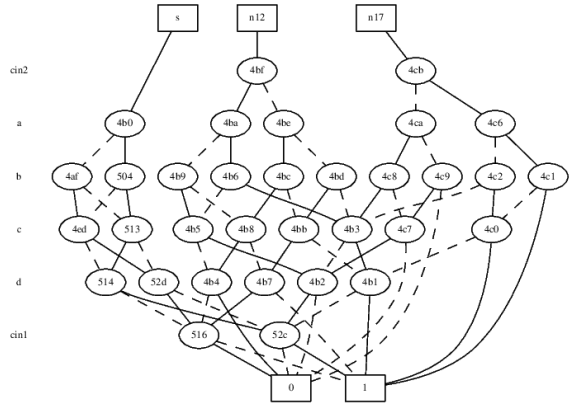
   By applying the command “bdd”, the “nd”, “edge”, and “lev” are still the same, the only different is that it has “bdd = 27” (below), which is the BDD of the node function. You can also notice that the structure of the graph is still nearly the same (right).



```
four_number_serial_adder      : i/o =    4/    1  lat =    2  nd =     3  edge =     18  bdd  =    27  lev = 1
```

   However, by applying the command ”collapse”, the global function is built using BDD, the graph is shown below and “bdd = 27”. (The # of latch will not change)

```
four_number_serial_adder      : i/o =    4/    1  lat =    2  nd =     3  edge =     17  bdd  =    27  lev = 1
```



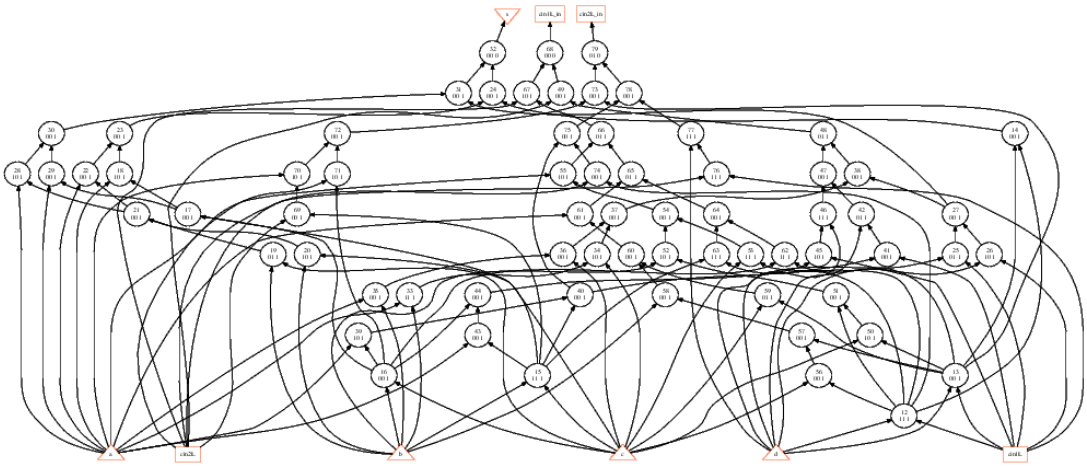
(b) Given a structurally hashed AIG, find a sequence of ABC commands to covert it to a logic network with node function expressed in sum-of-products (SOP).

Use the command “logic”.

The detail of the command is below,

logic – Transforms the AIG into a logic network with the SOP representation of the two-input AND-gates.

With the command, we can get the SOP representation shown as below,



- **Appendix:**

a	b	c	d	cin1	cin2	cout2	cout1	s
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	0
0	0	0	0	1	0	0	0	1
0	0	0	0	1	1	0	1	1
0	0	0	1	0	0	0	0	1
0	0	0	1	0	1	0	1	1
0	0	0	1	1	0	0	1	0
0	0	0	1	1	1	1	0	0
0	0	1	0	0	0	0	0	1
0	0	1	0	0	1	0	1	1
0	0	1	0	1	0	0	1	0
0	0	1	0	1	1	1	0	0
0	0	1	1	0	0	0	1	0
0	0	1	1	0	1	1	0	0
0	0	1	1	1	0	0	1	1
0	0	1	1	1	1	1	0	1
0	1	0	0	0	0	0	0	1
0	1	0	0	0	1	0	1	1
0	1	0	0	1	0	0	1	0
0	1	0	0	1	1	1	0	0
0	1	0	1	0	0	0	1	0
0	1	0	1	0	1	1	0	0
0	1	0	1	1	0	0	1	1
0	1	0	1	1	1	1	0	1
0	1	1	0	0	0	0	1	0
0	1	1	0	0	1	1	0	0
0	1	1	0	1	0	0	1	1
0	1	1	0	1	1	1	0	1
0	1	1	1	0	0	0	1	1
0	1	1	1	0	1	1	0	1
0	1	1	1	1	0	1	0	0
0	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	1
1	0	0	0	1	0	0	1	0

1	0	0	0	1	1	1	0	0
1	0	0	1	0	0	0	1	0
1	0	0	1	0	1	1	0	0
1	0	0	1	1	0	0	1	1
1	0	0	1	1	1	1	0	1
1	0	1	0	0	0	0	1	0
1	0	1	0	0	1	1	0	0
1	0	1	0	1	0	0	1	1
1	0	1	0	1	1	1	0	1
1	0	1	1	0	0	0	1	1
1	0	1	1	0	1	1	0	1
1	0	1	1	1	0	1	0	0
1	0	1	1	1	1	1	1	0
1	1	0	0	0	0	0	1	0
1	1	0	0	0	1	1	0	0
1	1	0	0	1	0	0	1	1
1	1	0	0	1	1	1	0	1
1	1	0	1	0	0	0	1	1
1	1	0	1	0	1	1	0	1
1	1	0	1	1	0	1	0	0
1	1	0	1	1	1	1	1	0
1	1	1	0	0	0	0	1	1
1	1	1	0	0	1	1	0	1
1	1	1	0	1	0	1	0	0
1	1	1	0	1	1	1	1	0
1	1	1	1	0	0	1	0	0
1	1	1	1	0	1	1	1	0
1	1	1	1	1	0	1	0	1
1	1	1	1	1	1	1	1	1

- **Reference:**

https://people.eecs.berkeley.edu/~alanmi/abc/