PA1

R14K41021 張仕謙

2. (a)

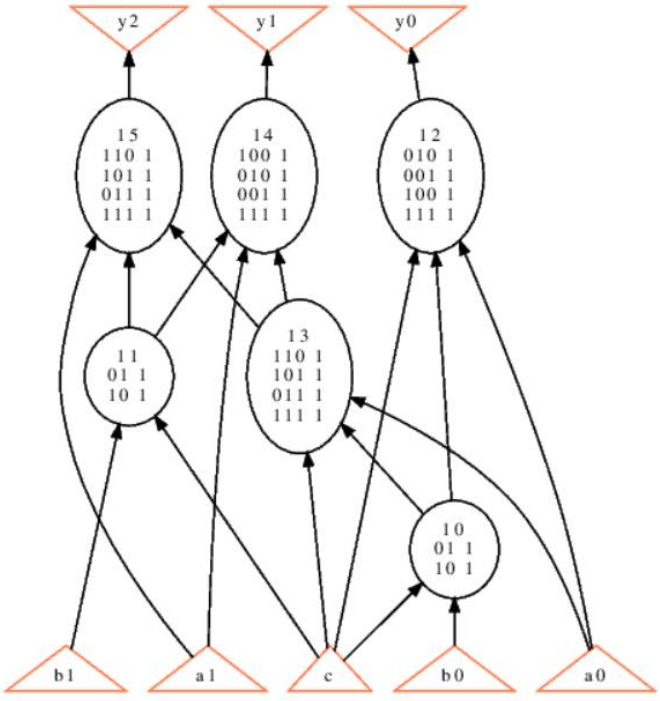
The ALU can be constructed as a 2-bit full adder by taking the input c as the carry-in for the LSB (c_0) of the adder and by checking whether B should be bitwise inverted. The reason is that when performing addition (i.e., c = 0), the carry-in for the LSB is $c_0 = c = 0$, so the actual operation is Y = A + B + 0. On the other hand, when performing arithmetic subtraction, Y = A – B (i.e., c = 1), we first apply the 2's complement to B, such that B becomes (~B + 1), and then perform addition. Under this structure, it is acceptable to construct the ALU as a 2-bit adder by using the carry-in for the LSB of the adder and preprocessing the input for B.

(b)

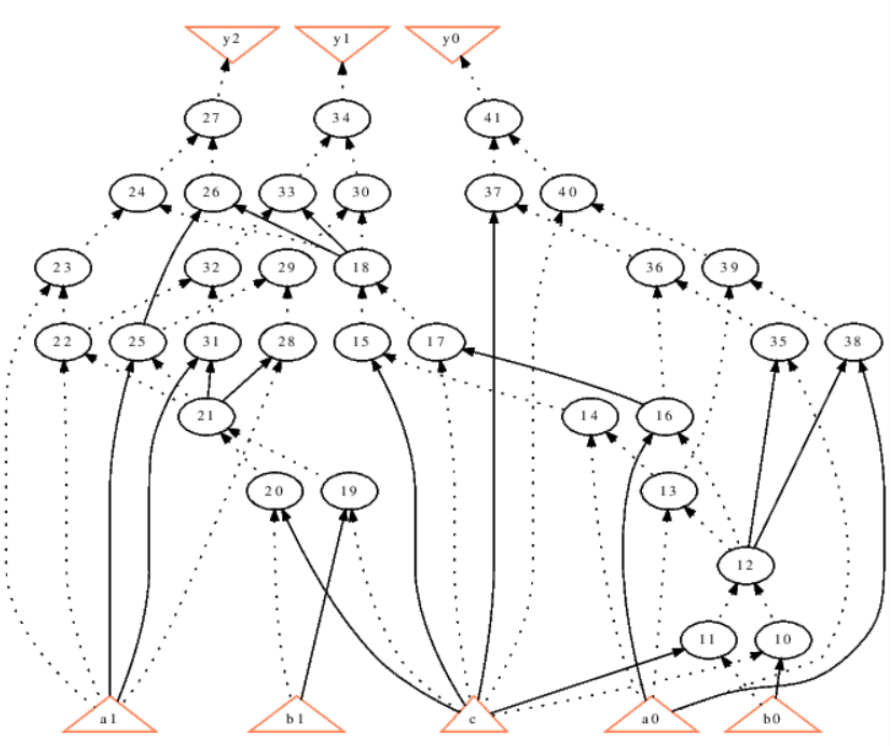
The following are my screenshot of the results after running the commands.

```
abc 01> read l
lib/ lsv/
abc 01> read lsv/pa1/alu_v2.blif
abc 02> show
abc 02> Warning: Missing charsets in String to FontSet conversion

abc 02> ps
alu                        : i/o =    5/    3  lat =    0  nd =     6  edge =     16  cube =    20  lev = 3
abc 02> strash
abc 03> show
abc 03> Warning: Missing charsets in String to FontSet conversion

abc 03> collapse
abc 04> show bdd -g
abc 04> Warning: Missing charsets in String to FontSet conversion
```

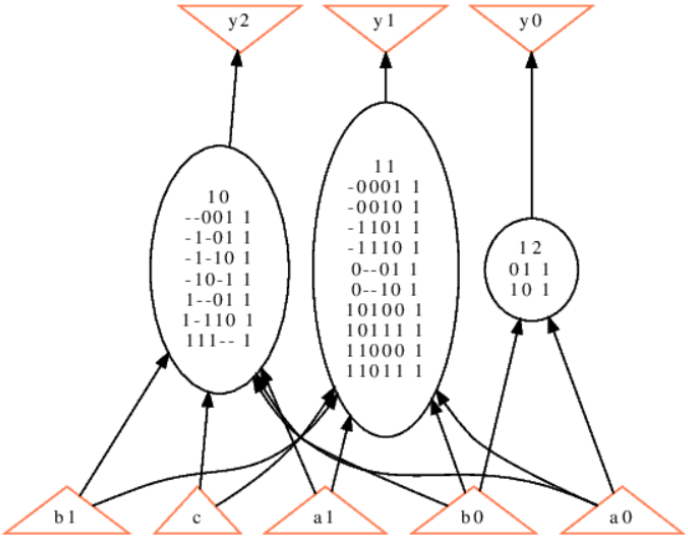
(1) Alu.blif show



(2) AIG show



(3) BDD show



3. (a)

```
abc 07> read lsv/pa1/alu.blif
abc 08> aig
abc 08> ps
alu                       : i/o =    5/    3  lat =    0  nd =     6  edge =     16  aig  =    36  lev = 3
abc 08> strash
abc 09> ps
alu                       : i/o =    5/    3  lat =    0  and =     32  lev =  8
abc 09> read lsv/pa1/alu.blif
abc 10> bdd
abc 10> ps
alu                       : i/o =    5/    3  lat =    0  nd =     6  edge =     16  bdd  =    18  lev = 3
abc 10> collapse
abc 11> ps
alu                       : i/o =    5/    3  lat =    0  nd =     3  edge =     12  bdd  =    18  lev = 1
```

(a) - 1.

The two descriptions of the command provided by -help:

Command “aig” converts node functions to AIG.

Command “strash” transforms combinational logic into an AIG.

Based on the above description, the following are my observations.

For “aig,” each node’s function is expressed as an AIG, but the original network structure is preserved. Therefore, network parameters such as the number of nodes, edges, and other statistics still exist. In comparison, “strash” rebuilds the combinational network into a single AIG and merges isomorphic subgraphs through structural hashing. As a result, only the number of AND nodes and the network level are reported in the statistics.

(a) - 2

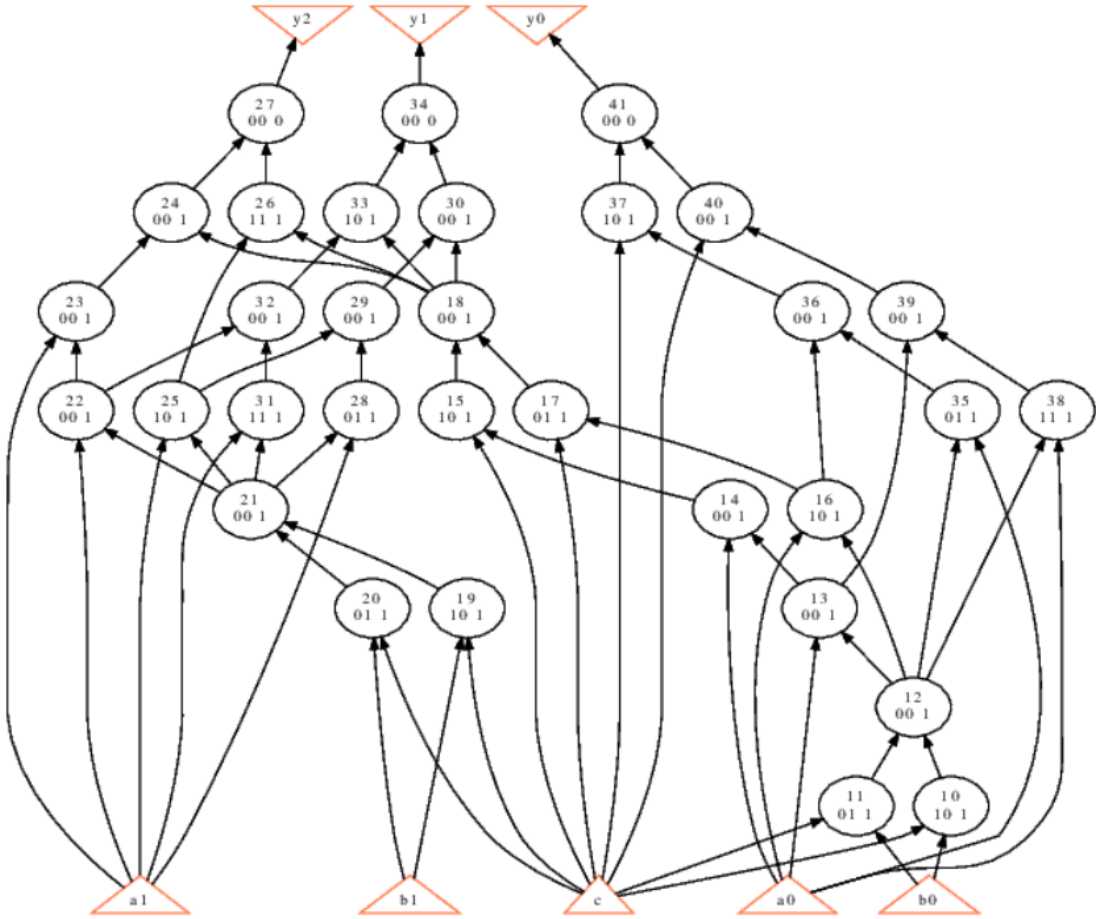
Command “bdd” converts node functions to BDD.

Command “collapse” collapses the network by constructing global BDDs

The same as with subproblem (a) - 1, for “bdd,” each node’s function is expressed as a BDD, but the original network structure is preserved. Therefore, network parameters such as the number of nodes, edges, and other statistics still exist. In comparison, “collapse” simplifies and factors the network, which halves the number of nodes and reduces the network depth.

3. (b)

```
abc 01> read lsv/pa1/alu.blif
abc 02> ps
alu                       : i/o =    5/    3  lat =    0  nd =     6  edge =     16  cube =    20  lev = 3
abc 02> strash
abc 03> ps
alu                       : i/o =    5/    3  lat =    0  and =     32  lev =  8
abc 03> logic
abc 04> ps
alu                       : i/o =    5/    3  lat =    0  nd =    32  edge =     64  cube =    32  lev = 8
abc 04> show
```


y2
y1
y0
27
00 0
34
00 0
41
00 0
24
00 1
26
11 1
33
10 1
30
00 1
37
10 1
40
00 1
23
00 1
32
00 1
29
00 1
18
00 1
36
00 1
39
00 1
22
00 1
25
10 1
31
11 1
28
01 1
15
10 1
17
01 1
35
01 1
38
11 1
21
00 1
14
00 1
16
10 1
20
01 1
19
10 1
13
00 1
12
00 1
11
01 1
10
10 1
a1
b1
c
a0
b0