Logic Synthesis & Verification, Fall 2024
National Taiwan University

Programming Assignment 1

Sahil Singh | R13943171

2 [Using ABC]

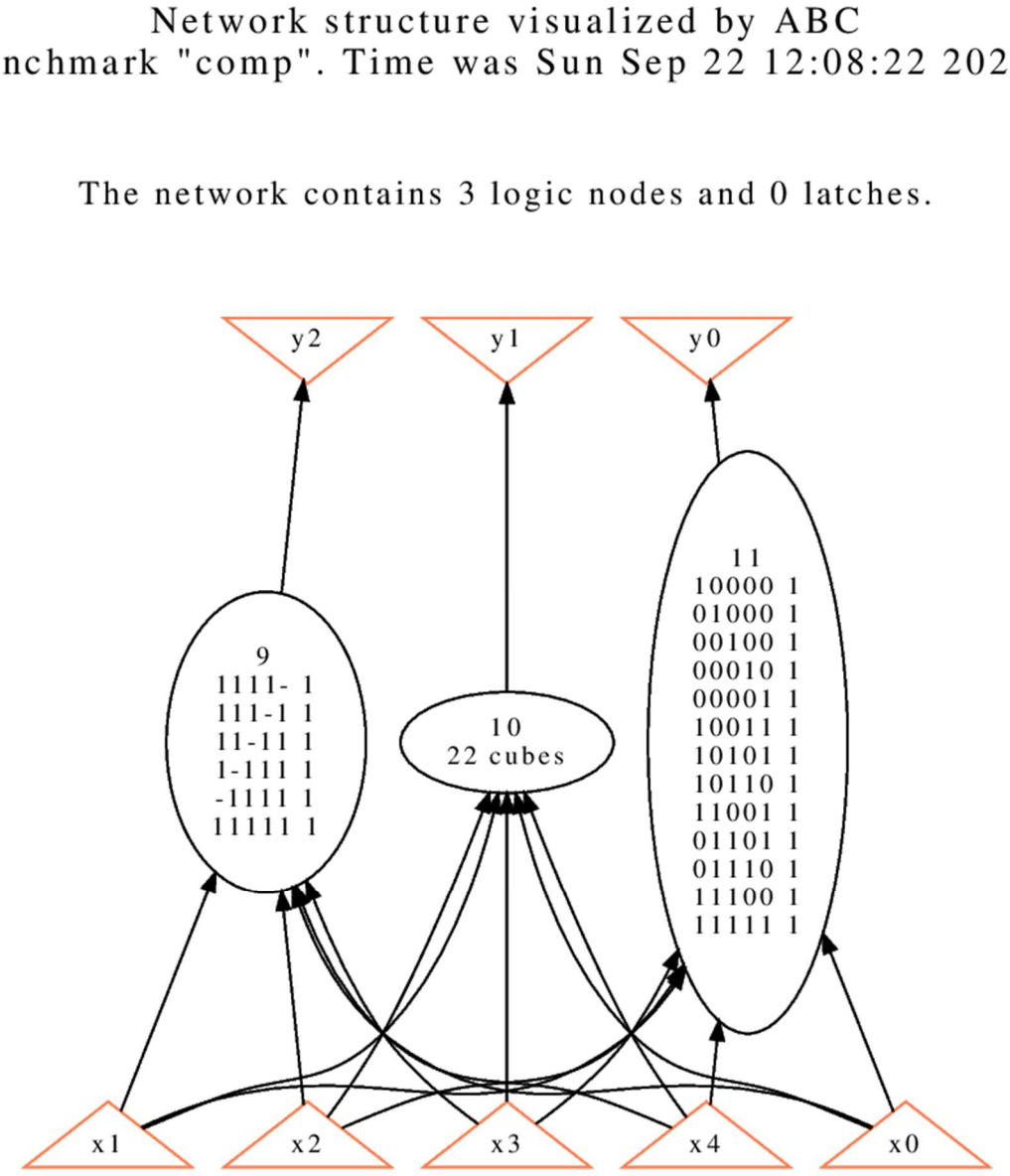
(a) Implemented the logic for counting the number of 1s in the 5 inputs and outputs the 3-bit unsigned integer representation. (comp.blif)

(b) 1. read command

2. print_stats command

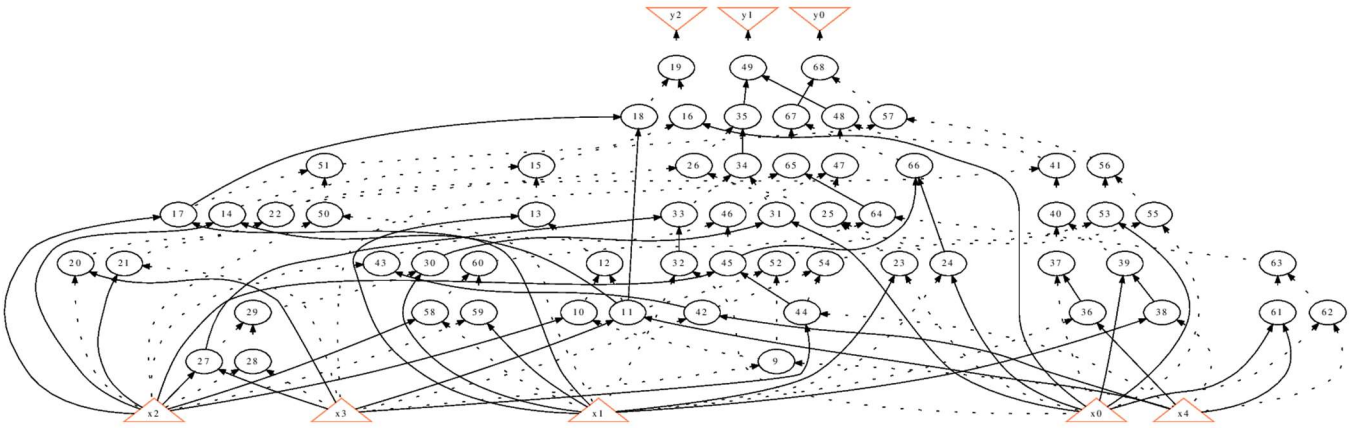
```
================================================
abc 01> read comp.blif
abc 02> print_stats
comp                        : i/o =    5/    3  lat =    0  nd =     3  edge =     15  cube =    41  lev = 1
abc 02>
```

3. show command

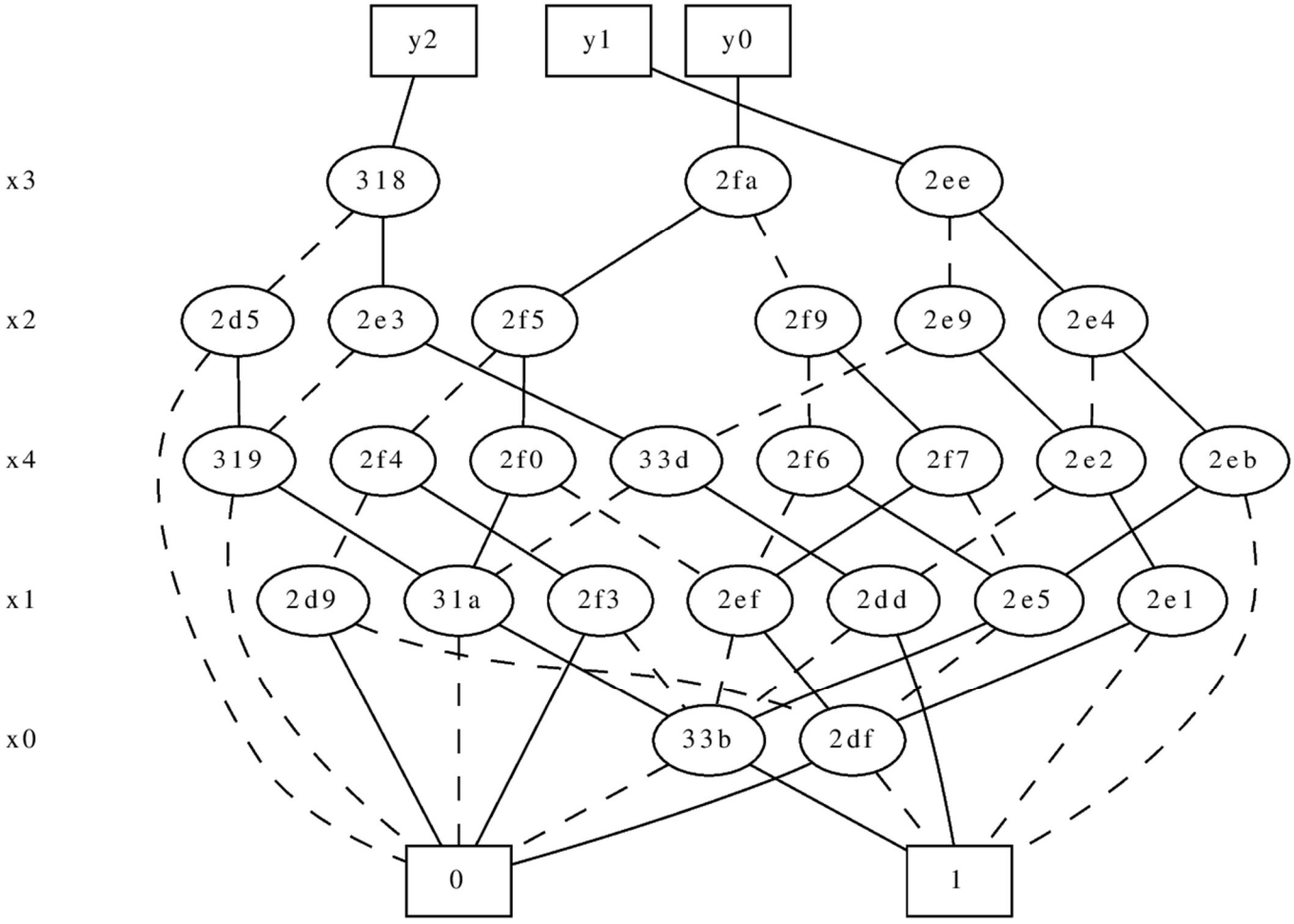


4. strash command

5. show command



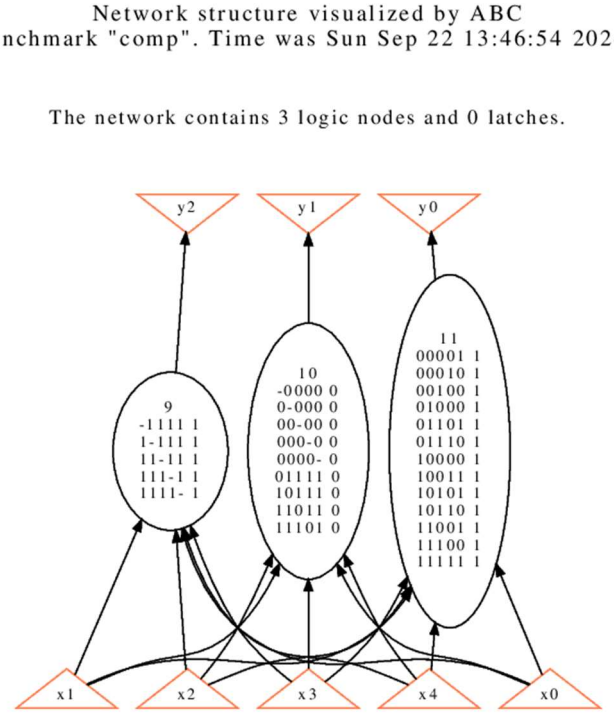
6. collapse command
7. show_bdd -g command



3 [ABC Boolean Function Representations]

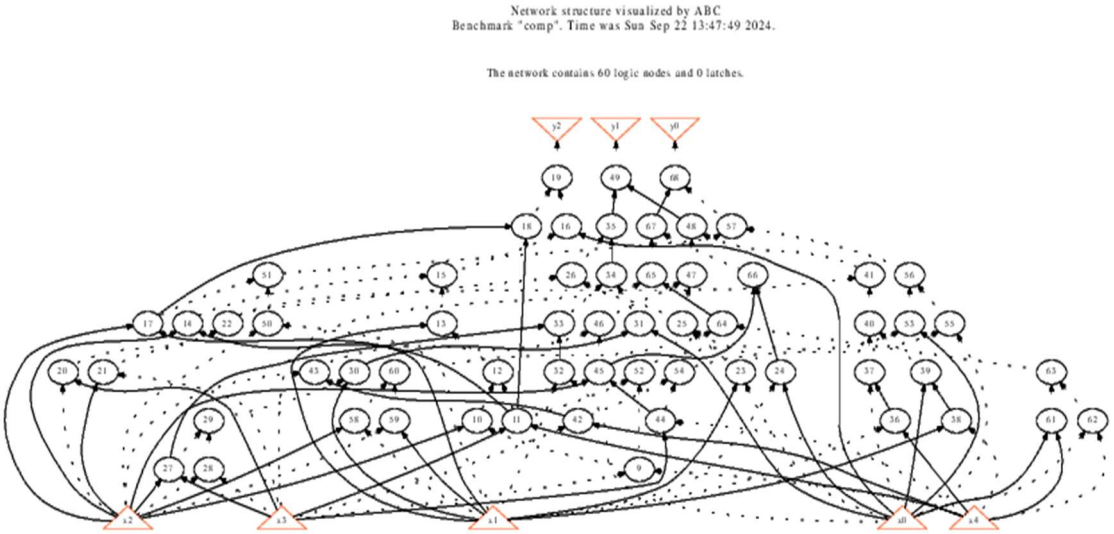
(a) (1) AIG (And-Inverter Graph):

The AIG representation shows a logic network as a combination of AND gates and inverters. Each node represents a logic operation, and connections between them represent dependencies. In an AIG, the main components are AND nodes, and optional inverters can be applied to the inputs. When we use the aig command on comp.blif, the result is a circuit converted to AIG format, showing how the inputs and outputs of the compressor are organized using a network of AND gates and inverters.



Structurally Hashed AIG (strash):

The structurally hashed AIG (obtained by the strash command) simplifies the AIG by reusing identical subgraphs. It eliminates redundant nodes and removes dangling (unused) AIG nodes, resulting in a more efficient representation. Structurally hashing makes the circuit more compact and can optimize performance by reducing the number of nodes.



The main difference between AIG and structurally hashed AIG is in the number of nodes and simplification of the logic. Structurally hashed AIG will generally have fewer nodes due to the elimination of redundant logic and unused nodes.

```
abc 01> read comp.blif
abc 02> print_stats
comp                          : i/o =    5/    3  lat =    0  nd =     3  edge =     15  cube =    41  lev = 1
abc 02> aig
abc 02> print_stats
comp                          : i/o =    5/    3  lat =    0  nd =     3  edge =     15  aig  =    67  lev = 1
```

Initial print_stats after reading comp.blif

- i/o: Inputs (5) and outputs (3), as defined in the 5-to-3 compressor circuit.
- lat: Latch count (0), indicating the circuit is purely combinational (no sequential elements like flip-flops).
- nd: Node count (3), which shows that there are only 3 logic gates used in the network.
- edge: Number of edges (15), representing the number of connections between the logic gates.
- cube: Number of product terms in the sum-of-products representation (41).
- lev: Logic depth or number of levels in the circuit (1). The depth is low, meaning the circuit has minimal delays.

After running aig command

The circuit has been converted into an And-Inverter Graph (AIG). The aig keyword in the stats shows the number of AIG nodes (67). This means that while the circuit still has 3 primary nodes in terms of logical gates, AIG creates additional internal nodes for the AND gates and inverters. lev remains at 1, indicating that the circuit's logic depth hasn't changed from the original. This means that the AIG doesn't introduce any additional levels of logic.

```
abc 02> strash
abc 03> print_stats
comp                          : i/o =    5/    3  lat =    0  and =     60  lev =  7
abc 03> show
```

After running strash command

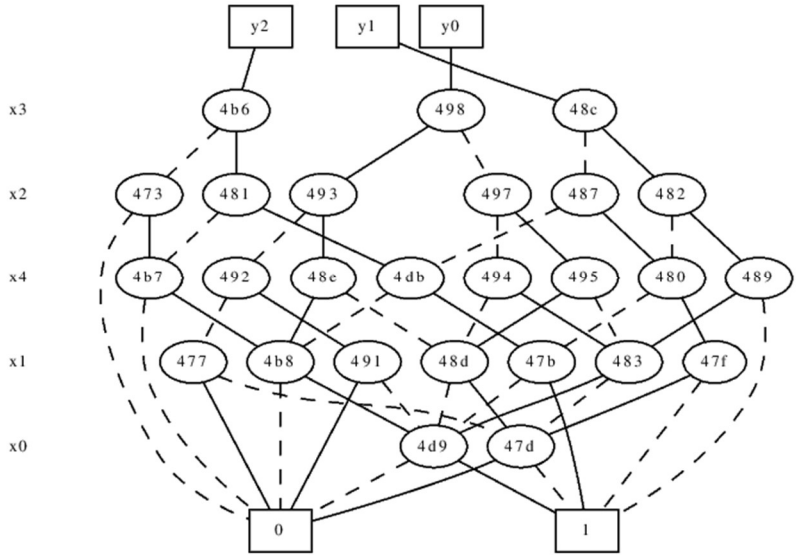
After running structural hashing (strash), the circuit is now optimized. The number of AND nodes (and = 60) is reduced from the earlier 67 nodes in the AIG version. The logic depth (lev = 7) has increased to 7 levels, indicating that structural hashing has reorganized the circuit into a more optimized configuration with deeper logic but fewer nodes.

The initial aig command creates a representation of logic network in terms of AND gates and inverters, adding internal nodes (67 AIG nodes). After strash, the circuit becomes more optimized, reducing the number of nodes (60 AND nodes) and potentially improving efficiency. However, this increases the circuit depth (from 1 to 7).

(2) BDD (Binary Decision Diagram):

A BDD represents the logic as a decision tree, where each node corresponds to an input variable, and paths represent the evaluation of the function based on different input combinations. It provides a compact representation for Boolean functions and can often simplify certain types of logic.

Running the bdd command on "comp.blif" converts the logic network into a BDD format, showing how the compressor's output is determined by different combinations of inputs.



Collapsed BDD (collapse):

A collapsed BDD (produced by the collapse command) further optimizes the BDD by constructing global BDDs for the entire network. It may reorder variables and remove redundant nodes, leading to an even more compact representation. The collapsing process often results in a more efficient structure, especially for large circuits with repetitive subfunctions.

```
abc 02> bdd
abc 02> print_stats
comp                          : i/o =    5/    3  lat =    0  nd =     3  edge =     15  bdd  =    29  lev = 1
abc 02> show
```

This converts the logic network into a Binary Decision Diagram (BDD) representation. The bdd = 29 indicates there are 29 BDD nodes created for this representation. lev remains at 1, indicating the same logic depth as the original circuit. The transformation from the basic logic network to BDD doesn't add any extra depth, but it introduces an alternative representation for easier manipulation of Boolean functions.

After running collapse command

```
abc 02> collapse
abc 03> print_stats
comp                          : i/o =    5/    3  lat =    0  nd =     3  edge =     15  bdd  =    29  lev = 1
abc 03> show
```

The collapse command constructs global Binary Decision Diagrams (BDDs) for the circuit and simplifies it. In this case, the number of BDD nodes remains the same at 29, meaning no further reduction is achieved through collapsing. lev remains at 1, indicating that collapsing doesn't increase or decrease the logic depth

for this particular circuit. The network remains relatively simple, as seen from the constant depth.

The command bdd creates a Binary Decision Diagram with 29 nodes. This is an alternative representation of the logic network, useful for Boolean function manipulation. Running collapse doesn’t reduce the BDD node count further (remains 29). This suggests that the logic network in its BDD form is already minimal and cannot be simplified further.

(b) To convert a structurally hashed AIG to a logic network with node functions expressed in sum-of-products (SOP), use the following sequence of ABC commands: read, strash, and logic.

```
abc 01> read comp.blif
abc 02> print_stats
comp                          : i/o =    5/    3  lat =    0  nd =     3  edge =     15  cube =    41  lev = 1
abc 02> logic
Error: This command is only applicable to strashed networks.
abc 02> strash
abc 03> logic
abc 04> print_stats
comp                          : i/o =    5/    3  lat =    0  nd =    60  edge =    120  cube =    60  lev = 7
abc 04> show
```