Programming Assignment 1

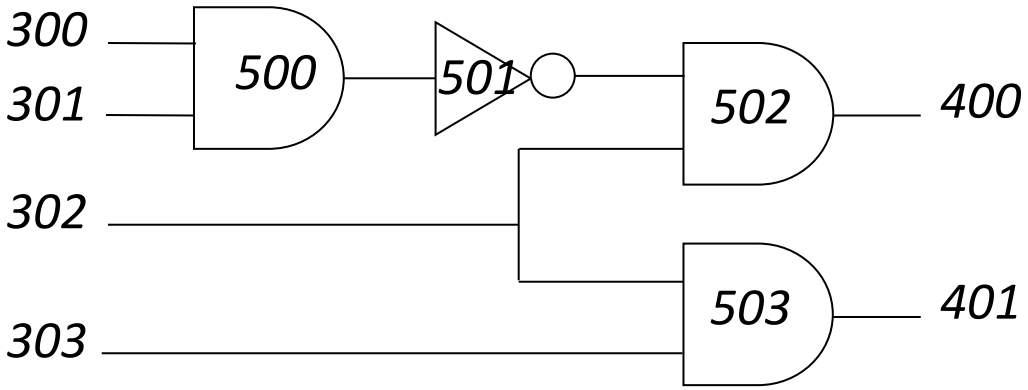
A manually created netlist parser

Introduction

- Netlist (net-list)
  - A format of a circuit in the gate-level representation
- Objective
  - Write a program to
    - **read** a circuit described by a pre-defined netlist
    - **simulate** the circuit for two input patterns
      - all primary inputs are 0 and 1, respectively
    - **output** simulation results

Combinational circuits

- A combinational circuit is composed of primary inputs, primary outputs, and logic gates



- Primary inputs: *a*, *b*, *c*, and *d*
- Primary outputs: *x* and *y*
- Logic gates: *n6*, *n7*, *n8*, and *n9*

Netlist format

- The are four parts
  - Comments
  - Primary inputs
  - Primary outputs
  - Logic gates

```
# 4 inputs  -- input_data(4)
# 2 outputs -- output_signal(2)
# 6 gates ( 2 BUFFs + 1 NOT + 3 ANDs )

INPUT(300)
INPUT(301)
INPUT(302)
INPUT(303)

OUTPUT(400)
OUTPUT(401)

400 = BUFF(502)
401 = BUFF(503)

501 = NOT(500)

500 = AND(300, 301)
502 = AND(501, 302)
503 = AND(302, 303)
```

Comments

- In the beginning of the file, a "#" begins a comment that extends to the end of the current line
- Comments give a brief description of the circuit

```
# 4 inputs  -- input_data(4)
# 2 outputs -- output_signal(2)
# 6 gates ( 2 BUFFs + 1 NOT + 3 ANDs )

…
```

Primary inputs

- "INPUT($signal_id)" indicates a primary input, where "$signal_id" is the signal identifier of the primary input
- Every signal identifier is an integer
- Each primary input is listed in one line

```
…

INPUT(300)
INPUT(301)
INPUT(302)
INPUT(303)

…
```

Primary outputs

- "OUTPUT($signal_id)" indicates a primary output, where "$signal_id" is the signal identifier of the primary output
- Every signal identifier is an integer
- Each primary output is listed in one line

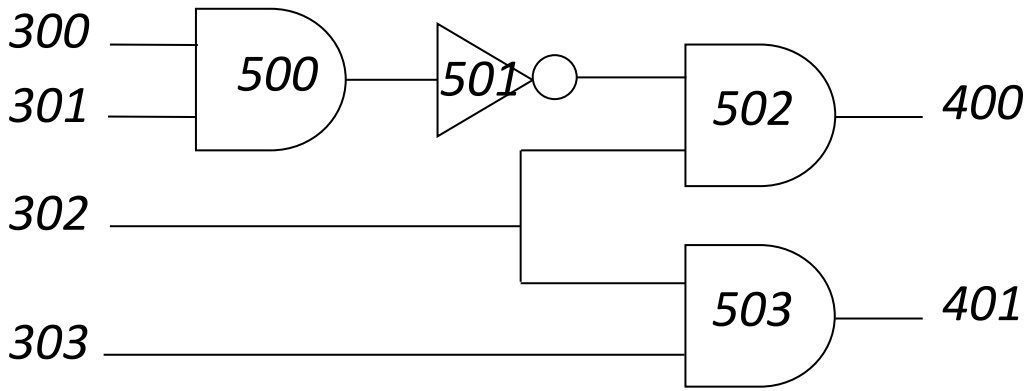
```
…

OUTPUT(400)
OUTPUT(401)

…
```

Logic gates

- There are 8 types of logic gates as follows
  - #signal_id1 = BUFF($signal_id2)
  - #signal_id1 = NOT($signal_id2)
  - #signal_id1 = AND($signal_id2, $signal_id3)
  - #signal_id1 = NAND($signal_id2, $signal_id3)
  - #signal_id1 = OR($signal_id2, $signal_id3)
  - #signal_id1 = NOR($signal_id2, $signal_id3)
  - #signal_id1 = XOR($signal_id2, $signal_id3)
  - #signal_id1 = NXOR($signal_id2, $signal_id3)
- #signal_id1: output signal
- #signal_id2 and #signal_id3: input signals
- Every signal identifier is an integer
- Each gate is described in one line

Example

```
# 4 inputs  -- input_data(4)
# 2 outputs -- output_signal(2)
# 6 gates ( 2 BUFFs + 1 NOT + 3 ANDs )

INPUT(300)
INPUT(301)
INPUT(302)
INPUT(303)

OUTPUT(400)
OUTPUT(401)

400 = BUFF(502)
401 = BUFF(503)

501 = NOT(500)

500 = AND(300, 301)
502 = AND(501, 302)
503 = AND(302, 303)
```



Requirement

- In addition to the objective, your program should be executable on the Moon workstation (moon.cse.yzu.edu.tw) with the following method:

  >> ./a.out < design_00.isc

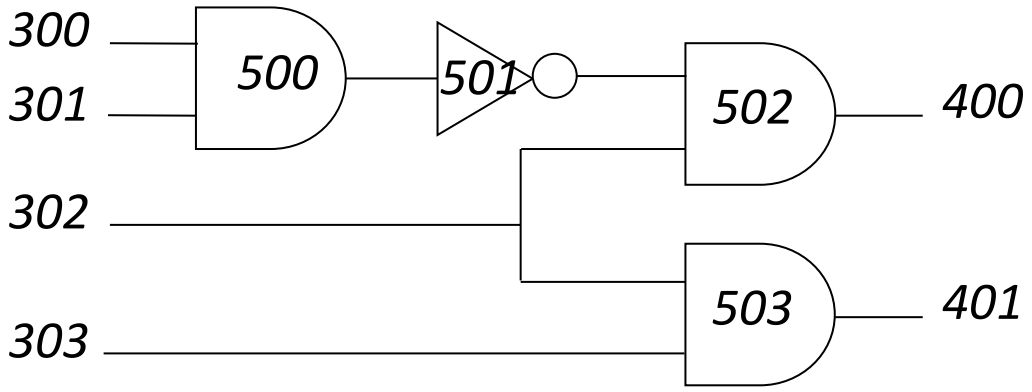
  Where a.out is the execution file and design_00.isc is the input file

Output of your program

- Output values of the two input patterns
  - All primary inputs are 0 and 1, respectively
- For example

  00

  01



Two issues you need to address

- How to store a combinational circuit
  - Directed acyclic graph
- How to simulate a combinational circuit
  - For a gate, before computing the output value, you need to compute all the input values
  - Thus, you need to find the topological sort order of the circuit, before you simulate the circuit

Delivery

- Your source code
- A readme describing how to compile and execute your code
- The execution results obtained by PrintScr for processing all the given benchmarks
- Upload all the files to Portal by 10/3(Tue.)
  - No late delivery is allowed