GEORGIA INSTITUTE OF TECHNOLOGY, ECE

Logic Simulator

September 29, 2015

Arunprasanna Sundararjan Poorna

Contents

1 INTRODUCTION

The objective of this project is to simulate logic gates efficiently. All gates have been pre processed and reduced to two input gates, thus making it easy to parse and simulate each gate. We maintain a stack of gates that can be evaluated and for each gate that is evaluated we check if the output of the gate translates to any of the inputs to other gates. This report describes the data structures used and provides the results for the inputs given to the simulator.

2 DATA STRUCTURES USED

This section describes the data structures used to implement the logic simulator. The project has been done in Python 2.7.X, and I have described the data structures in detailed below:

- There are several parameters stored as integer variables and these are: number of gates, number of nets, input nets, output nets etc.
- A class has been defined to wrap the variables required to define a gate and it's methods. This class is called *gates*
  - The class has a string "gate" that stores what type of gate it is. It also has integer variables to denote the inputs and the outputs.
  - The class also has several methods. It has an overloaded constructor that can be called to create an object as and when the input file is being read.
  - The class also has methods to display the information about the gate and evaluate the gate itself.
- A Dictionary in Python is a resizable hash table that stores key,value pairs. It has O(1) lookup for elements.
- A dictionary has been used to store the gate number, gate objects pairs
- A dictionary has been used to store the net number, net value pairs.
- A dictionary has been used to map each net to a list of gates that it drives. So that this can be looked up to find which gates may have both inputs to be valid after evaluating a gate.
  - This dictionary is imported from collections module since the native dictionary is insufficient.
  - We need a special dictionary that can help us store a linked list of values for each key.
- Note that the dictionary(hash table) is a powerful data structure, and it's power lies in the flexibility of storing key, value pairs which can be of any type.
- A list has been used as a stack in Python by using the .append() and .pop() methods. The stack contains a stack of objects.

3 PSEUDO CODE

This section describes the pseudo code of the algorithm used for the logic simulator.

- Do a single pass of the file, to find the number of gates, the maximum number of nets and also store the input and output gates.
- Initialize the dictionaries used(refer above), with default constructors to store -1 in inputs and outputs to indicate that they are invalid entries.
- Read the file line by line, creating objects of type gate and storing them in a dictionary with the gate no. (line no.) as the key.
- The type of gate is read and the corresponding constructor is invoked to create the object. Input2 is not valid for gates that are INV or BUF.
- A bit vector of required input length(number of inputs) is taken as input.
- Display an error if the length of input is incorrect and re-prompt for correct input.
- The corresponding net values are assigned based on the bit vector input.
- All these bits are converted to boolean, since Python uses Boolean logic for all bit wise operators. This is done by the function bit_to_boolean(bit).
- A corollary of the above function is also defined to convert the final boolean output to bits. This function is called boolean_to_bit(boolean).
- Go through all the gate objects and push those gates that have valid bits in all their inputs(not -1) into a stack.
- While the stack is not empty.
  - Pop the gate object.
  - Display the contents of the object.
  - Evaluate the gate.
  - Go through all the values in the list of values contained in the dictionary that maps nets to the gates the are given as input to.
  - Go through the above value which corresponds to a gate and see if both the inputs of the gate are now valid.
  - Push it into the stack if the inputs are now valid.
- Convert the boolean output to bit output using the function boolean_to_bit(boolean)
- Print the final bit vector obtained.

4 Results

4.1 s27.txt

INPUT VECTORS	OUTPUT VECTORS
1110101	1001
0001010	0100
1010101	1001
0110111	0001
1010001	1001

4.2 s298f_2.txt

INPUT VECTORS	OUTPUT VECTORS
10101010101010101	00000010101000111000
01011110000000111	00000000011000001000
11111000001111000	00000000001111010010
11100001110001100	00000000100100100101
01111011110000000	11111011110000101101

4.3 s344f_2.txt

INPUT VECTORS	OUTPUT VECTORS
10101010101010101011111111	10101010101010101010101101
01011110000000111000000	00011110000000100001111100
11111000001111000111111	00011100000111011000111010
11100001110001100000000	00001101111001111111000010
01111011110000000111111	10011101111000001001000100

4.4 s349f_2.txt

INPUT VECTORS	OUTPUT VECTORS
10101010101010101011111111	10101010101010101101010101
01011110000000111000000	00011110000000101011110000
11111000001111000111111	00011100000111010001111100
11100001110001100000000	00001101111001110010001111
01111011110000000111111	10011101111000001010000100

5 Appendix: Code

```
import os
import glob
import sys
from collections import defaultdict

files = []
for file in glob.glob("*.txt"):
        files.append(file)

print files
def bit_to_boolean(bit):
        if "1" in bit:
                return True
        else:
                return False

def boolean_to_bit(boolean):
        if boolean:
                return "1"
        else:
                return "0"

class gates:
        def __init__(self):
                self.gate =-1
                self.input1 = -1
                self.input2 = -1
                self.output = -1
        def __init__(self, string, input1, output, input2=None):
                if(input2==None):
                        self.gate = string
                        self.input1 = input1
                        self.input2 = -1
                        self.output = output
                else:
                        self.gate = string
                        self.input1 = input1
                        self.input2 = output
                        self.output = input2

        def display(self):
                print "Gate is: " + self.gate
                print "Inputs are",
                print self.input1, self.input2
                print "Output is:",
                print self.output
        def evaluate(self, nets):
                if "AND" in self.gate:
                        nets[self.output] = nets[self.input1] and nets[self.input2]
                if "OR" in self.gate:
```

```
                        nets[self.output] = nets[self.input1] or nets[self.input2]
                if "NAND" in self.gate:
                        nets[self.output] = not(nets[self.input1] and nets[self.input2])
                if "NOR" in self.gate:
                        nets[self.output] = not(nets[self.input1] or nets[self.input2])
                if "INV" in self.gate:
                        nets[self.output] = not(nets[self.input1])
                if "BUF" in self.gate:
                        nets[self.output] = nets[self.input1]

for file_ in files[1:2]:
        lines = []
        print "\n\n"
        print file_
        print "\n\n"
        with open(file_) as inputfile:
                for line in inputfile:
                        lines.append(line.split("\r\n")[0])
        lines = lines[:len(lines)]
        number_of_gates =0
        number_of_nets = 0
        inputs =[]
        outputs =[]
        both_input_list = ["AND", "OR", "NAND", "NOR"]
        single_input_list = ["INV", "BUF"]
        for index, line in enumerate(lines):
                if "INPUT" in str(line):
                        number_of_gates = index
        for line in lines:
                if "INPUT" in str(line):
                        inputs= (str(line).split()[1:len(str(line).split())-1])
        for line in lines:
                if "OUTPUT" in str(line):
                        outputs= (str(line).split()[1:len(str(line).split())-1])
        max_lines = map(int,lines[number_of_gates-1].split()[1:])
        number_of_nets= max(max_lines)
        nets ={}
        nets_inputgates =defaultdict(list)
        for i in range(1, number_of_nets+1):
                nets[i] = -1
        print "Gates␣#␣:",
        print number_of_gates
        dict_of_gates ={}
        for i in range(number_of_gates):
                if len(lines[i].split()) == 3:
                        a = gates(lines[i].split()[0], int(lines[i].split()[1]), int(lines[i].split
                        dict_of_gates[i+1]=a
                        nets_inputgates[int(lines[i].split()[1])].append(i+1)
                if len(lines[i].split()) ==4:
                        b = gates(lines[i].split()[0], int(lines[i].split()[1]), int(lines[i].split
                        dict_of_gates[i+1]=b
                        nets_inputgates[int(lines[i].split()[1])].append(i+1)
```

```
                nets_inputgates[int(lines[i].split()[2])].append(i+1)
inputs = map(int, inputs)
outputs = map(int, outputs)
print inputs
print outputs
print "Please␣provide␣the␣bit␣vector␣of␣length:",
print len(inputs)
bit_vector = raw_input()
if len(bit_vector) != len(inputs):
        print "Please␣enter␣a␣bit␣vector␣of␣length:",
        print len(inputs)
        bit_vector = raw_input()
print bit_vector
input_vector = []
for char in bit_vector:
        input_vector.append(bit_to_boolean(char))
print input_vector
stack = []
for input_, char in zip(inputs, input_vector):
        nets[input_] = char
for gate_number, gate_object in dict_of_gates.iteritems():
        if gate_object.gate in both_input_list:
                if nets[gate_object.input1] !=-1 and nets[gate_object.input2] != -1 :
                        stack.append(gate_object)
        if gate_object.gate in single_input_list:
                if nets[gate_object.input1] != -1:
                        stack.append(gate_object)

while stack:
        gate_object = stack.pop()
        gate_object.display()
        gate_object.evaluate(nets)
        for values in nets_inputgates[gate_object.output]:
                check_object= dict_of_gates[values]
                if check_object.gate in both_input_list:
                        if nets[check_object.input1] != -1 and nets[check_object.input2] !=
                                stack.append(check_object)
                if check_object.gate in single_input_list:
                        if nets[check_object.input1] != -1:
                                stack.append(check_object)

final_bit = ""

for output in outputs:
        print output,
        print nets[output]
        final_bit = final_bit+ boolean_to_bit(nets[output])

print final_bit
```