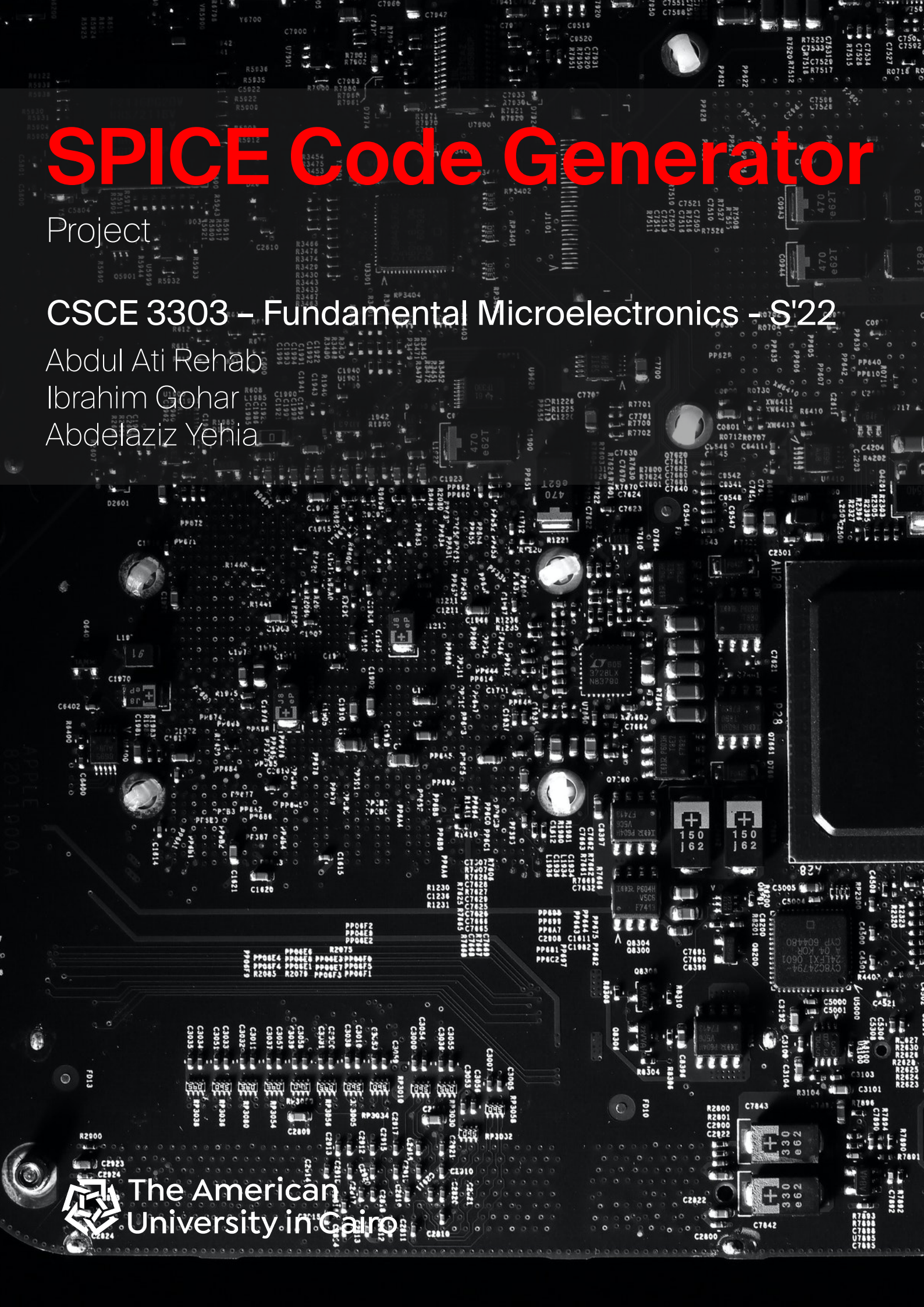
SPICE Code Generator

Project

CSCE 3303 – Fundamental Microelectronics – S'22

Abdul Ati Rehab
Ibrahim Gohar
Abdelaziz Yehia



The American University in Cairo

SPICE Code Generator

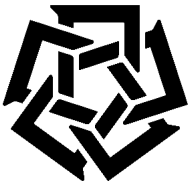
Project

by

Abdul Ati Rehab
Ibrahim Gohar
Abdelaziz Yehia

Student Name	Student ID
Rehab	900204245
Gohar	900203321
Yehia	900203361

Faculty: School of Science and Engineering, AUC
Instructor: Dr. Dalia Selim | Spring 2022



Preface

This report includes a short overview of a C++-based program that is used as a generator for CMOS circuits. The program takes some logic function(s) input from the user and generates the net-list code for the schematic of the SPICE circuit that is equivalent to the input logic function(s). Schematics as well as net-list codes are of high importance because they are dependencies that are required by simulators like SPICE upon which we depend to simulate complex circuits that are hard to be analyzed by hand.

Abdul Ati Rehab
Ibrahim Gohar
Abdelaziz Yehia
May 2022

Summary

This report represents a basic introduction to the C++ program that is attached with it. At the beginning, you can find a tutorial that introduces you to the program, the required dependencies required to run the program and the way it requires the input and returns the output. Next, you find a detailed description of the implementation methods used in the program including the logic, used algorithms and used data structures throughout the program. Afterwards, you find a description for the bonus features presented in the program and the way they were implemented. In the Test cases section, you find some test cases used to test the program and the output the program returns in those test cases. Finally, in the Appendices section, you find a copy of the program source code along with the Workload distribution throughout the project.

Contents

1

Introduction

The presented program is a C++ program that takes one or more Boolean expressions as input and outputs the corresponding SPICE netlist code for each of them. The user inserts the text (that is validated using a pattern that is in the HTML code of the web page) in the text field on the home page of the web site. Afterwards, they press the submit button to push the text to the CROW back end of the web site. The back end then calls the C++ program that is implemented and receives the output from that program. Finally, that output text is displayed in a white pop-up window that is activated using JavaScript upon hitting the ”Show Results” button.

2

How to use?

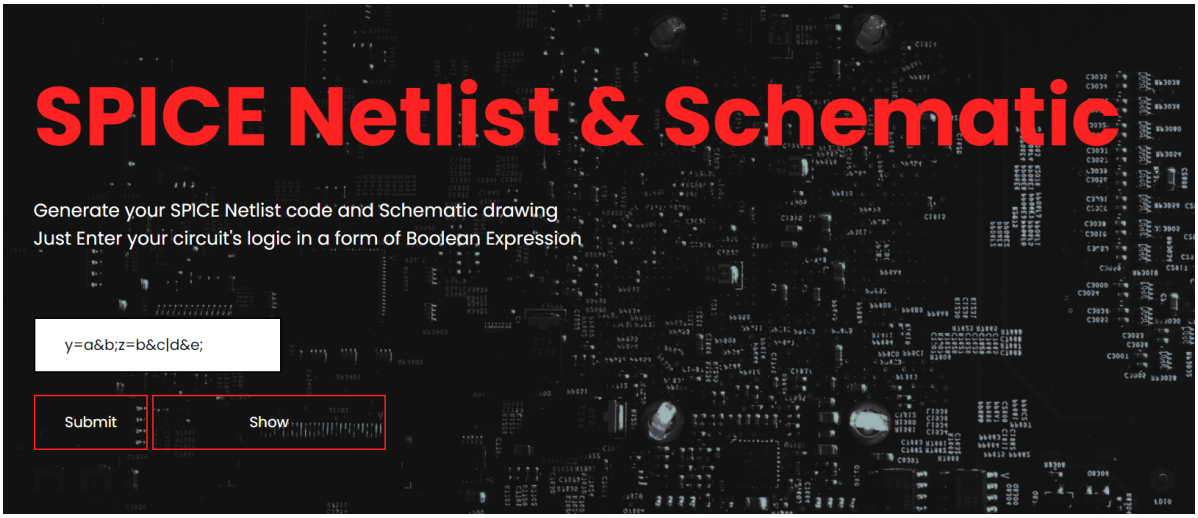
2.1. Dependencies and required libraries

- boost c++
- crow c++
- Ngrok

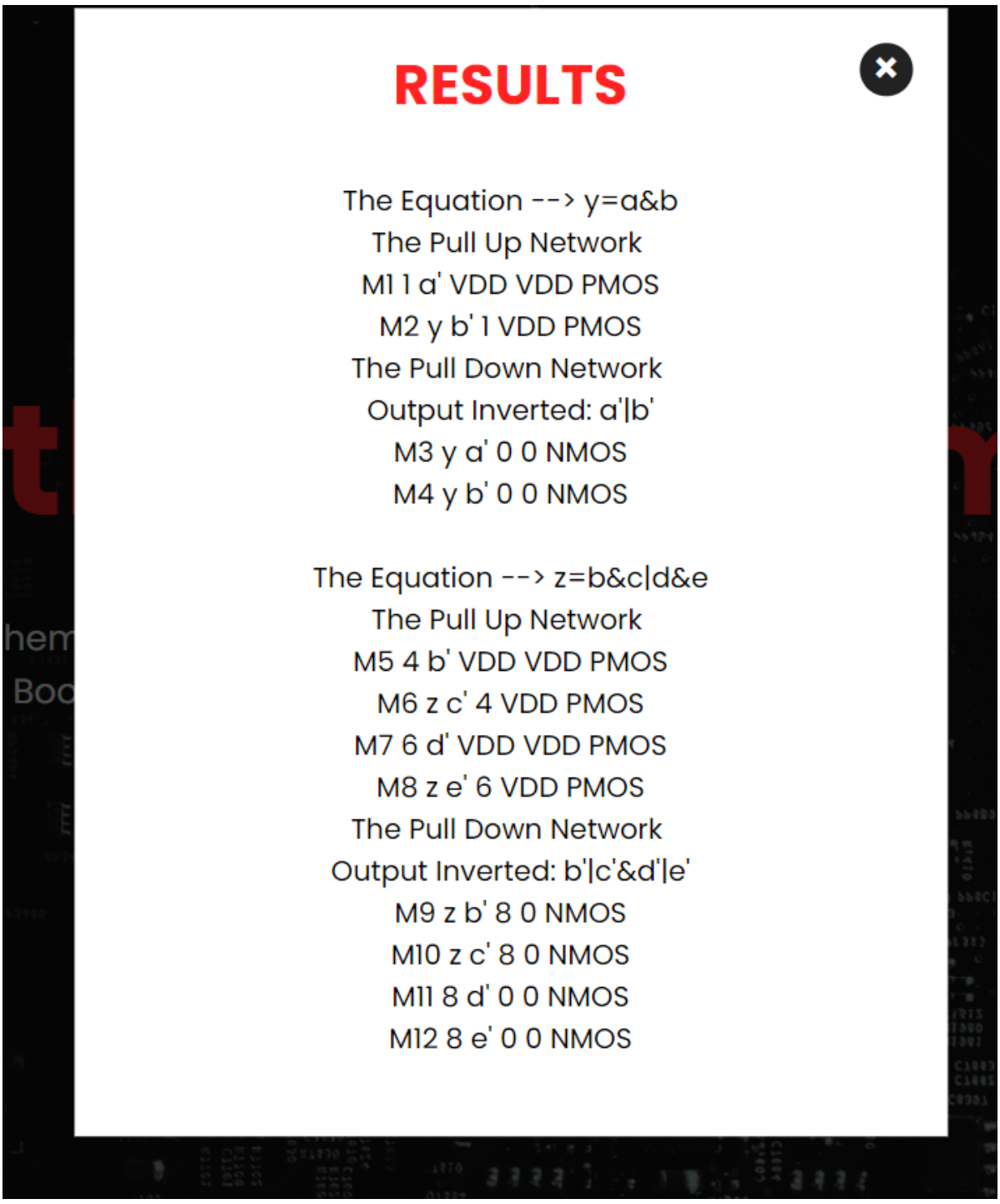
If you want to run it online and open it from your mobile phone, **the required statement to run it is** `Ngrok http 18080`

2.2. How to run?

The program is implemented as a Web-based application. You can find a friendly Graphical User interface that is easy to interact with. Simply, you can enter the required Boolean function(s) in the shown text box. Afterwards, you press the ”Submit” button to submit the results to the program. Finally, to show the results, you can press the ”Show” button. Here is a sample example.



As you can see, the input function(s) in the text box separating them using semi-colons. After wards, press the submit button to submit the input to the program. To show the results, press the Show button. The results of the previous input is shown below.


RESULTS
The Equation --> y=a&b
The Pull Up Network
M1 1 a' VDD VDD PMOS
M2 y b' 1 VDD PMOS
The Pull Down Network
Output Inverted: a'|b'
M3 y a' 0 0 NMOS
M4 y b' 0 0 NMOS
The Equation --> z=b&c|d&e
The Pull Up Network
M5 4 b' VDD VDD PMOS
M6 z c' 4 VDD PMOS
M7 6 d' VDD VDD PMOS
M8 z e' 6 VDD PMOS
The Pull Down Network
Output Inverted: b'|c'&d'|e'
M9 z b' 8 0 NMOS
M10 z c' 8 0 NMOS
M11 8 d' 0 0 NMOS
M12 8 e' 0 0 NMOS

3

Implementation

3.1. Program Flow

This section illustrated the flow of the program and the logic behind it, including the main algorithms employed as well as the data structure utilized. First, we take the input(s) from the user. Then, we call the functions pullUp() that produces the pull-up network, and we then call the function pullDown() that produces the pull-down network. We will illustrate the work flow of the program with the following example: y=a&b|c&d. Before, illustrating the logic of the pull-up and the pull-down network, we note that we have a defined struct named CMOS that has the data name, drain, gate, source, body (the exact same order to be outputted). The CMOS has a default constructuor and a parameterized one for taking the values from the user. We also have a function named demorgan() for inverting any boolean expression taken from the user. In our example, a&b|c&d is the input to the function in our example, and the output is a'|b'&c'|d'. Although we did not user parenthesis throughout our program, it is now understood the priority is reversed; now, we prioritize the OR '|' over the and '&'. This is equivalent to the parenthetical notation (a'|b')&(c'|d'). Now, let us illustrate the working of the pull-up and the pull-down networks.

3.1.1. The Pull-Up Network

For the pull-up network, we first take the string named logicFunction that contains the boolean function entered (only the right hand side of the equation useful for us, a&b|c&d in our case), and the output character (y in this example). The string is then split by the OR character '|' to prioritize working with "AND" first, and then make the "OR". We insert these splitted strings into the vector boolFunctions. Now, boolFunction is a vector of strings, where each string is a boolean function that will be evaluated first, and then will be ORed with other strings (boolFunctions = {"a&b","c&d"} in the example). Then, we further split each string in the vector boolFunctions by the character '&', to identify every input, and push that into a vector of vector of strings named inputs. Now, inputs = {{"a","b"},{"c","d"}}. Then, we invert every string input by calling the function invertInputs(). Now, inputs = {{"a'","b'"},{"c'","d'"}}. Then, we loop over all the string elements in inputs and check the following conditions:

1. If (for every vector in the inputs vector of vectors) we have only one input, its source will be connected to "VDD", its drain will be connected to the output character 'y' in this case, and its gate terminal will be connected to inputs[i][j].
2. Else if, for every first input in every vector of vectors which are (a') and (c') in our example, the corresponding transistor will have its source connected to "VDD" and its gate connected to inputs[i][j] = "a'" or "c'" in our example, and its drain will be connected to a certain terminal that will be given a unique number by the program.
3. Else if, for every last input in every vector of vectors which are (b') and (d') in our example, we will connect its source to a certain terminal given a unique number such that this terminal will be connected with the drain of the above series transistor (if any), and we will connect its drain with the output character which is "y" in our example, and of course the gate terminal will be connected with inputs[i][j] = "b'" or "c'" in the example.

4. Else, the input is intermediate (not the first and not the last), we will have its source (also its drain) connected to a certain terminal such that this terminal is connected to the drain of its above transistor (also connected to the source of its lower transistor), and its gate will be connected of inputs[i][j].

For every case of the above cases, we have the type of the transistor as ”PMOS” and the body is always connected to ”VDD”

3.1.2. The Pull-Down Network

For the pull-down network, much of what is said in the pull-up network is still applied with some minor modifications. We still take the string named logicFunction; however, this time we call the function demorgan and convert the input from ”a&b|c&d” to ”a’|b’&c’|d’”. Now, as we illustrated above, we will prioritize the ”OR” first then the ”AND”. Therefore, the string is split by the ”AND” character ’&’. We insert these splitted strings into the vector boolFunctions. Now, boolFunction is a vector of strings, where each string is a boolean function that will be evaluated first, and then will be ANDed with other strings (boolFunctions = {”a’|b’”,”c’|d’”} in the example). Then, we further split each string in the vector boolFunctions by the character ’|’, to identify every input, and push that into a vector of vector of strings named inputs. Now, inputs = {{”a’”,”b’”},{”c’”,”d’”}}. However, unlike what we have done in the pull-up network, we do not inver the inputs, so we will keep it as they are. Now, inputs = {{”a’”,”b’”},{”c’”,”d’”}}. Then, we loop over all the string elements in inputs and check conditions very similar to the conditions in the pull-up network, but rather applied on every parallel part not on individual inputs.

3.2. Assumptions

1. There is an assumption made in our project is that we use the inverted inputs as if they were given.In the above mentioned example for instance, we use a’ directly without illustrating how a’ was obtained from a. This is because it is known that if we were to invert an input, we would have to use two transistor one PMOS for the pull-up network and another one for the pull-down network. For example, a’ would be realized as follows:

$$M1\ a'\ a\ vdd\ vdd\ PMOS$$

$$M2\ a'\ a\ 0\ 0\ NMOS$$

However, for the sake of minimzing redundancy, we skipped this step of describing the circuit that realizes every inverter.

4

Bonus Features

4.1. GUI & Web-based Application

The program has been deployed as a web-based application. The front-end part is developed using **HTML and CSS**. It is a user friendly screen with an input field for the user to enter the desired boolean expression and two buttons, one for the sending the boolean expression to the back-end part and the other to get the result from the back-end. The program has been locally hosted to check it's functionality using Crow. **Crow** is microframework for running web services that uses c++ as a back-end for its applications. Then, the application was tested online on the network using **Ngrok**, a cross-platform application that exposes local server ports to the Internet.

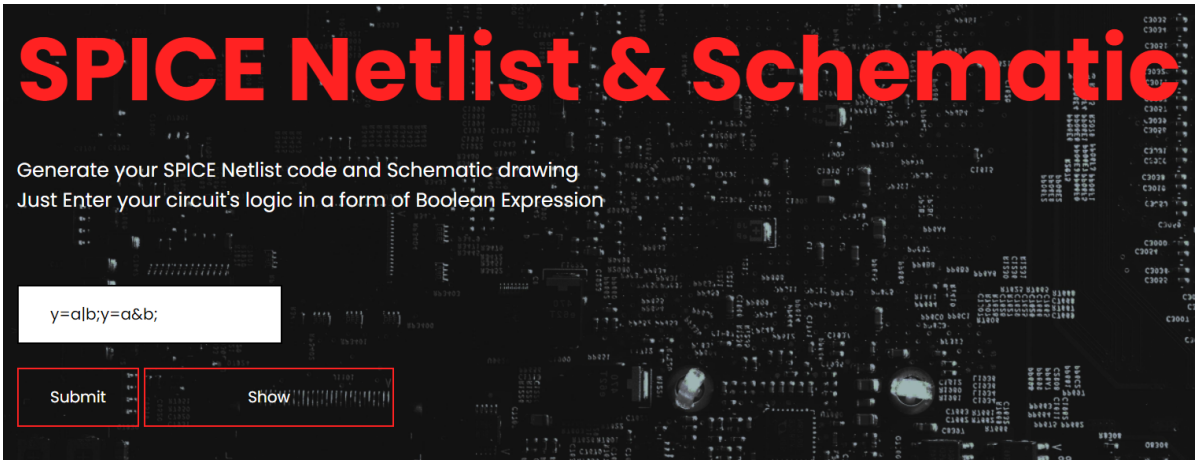
4.2. Multi-input Boolean functions

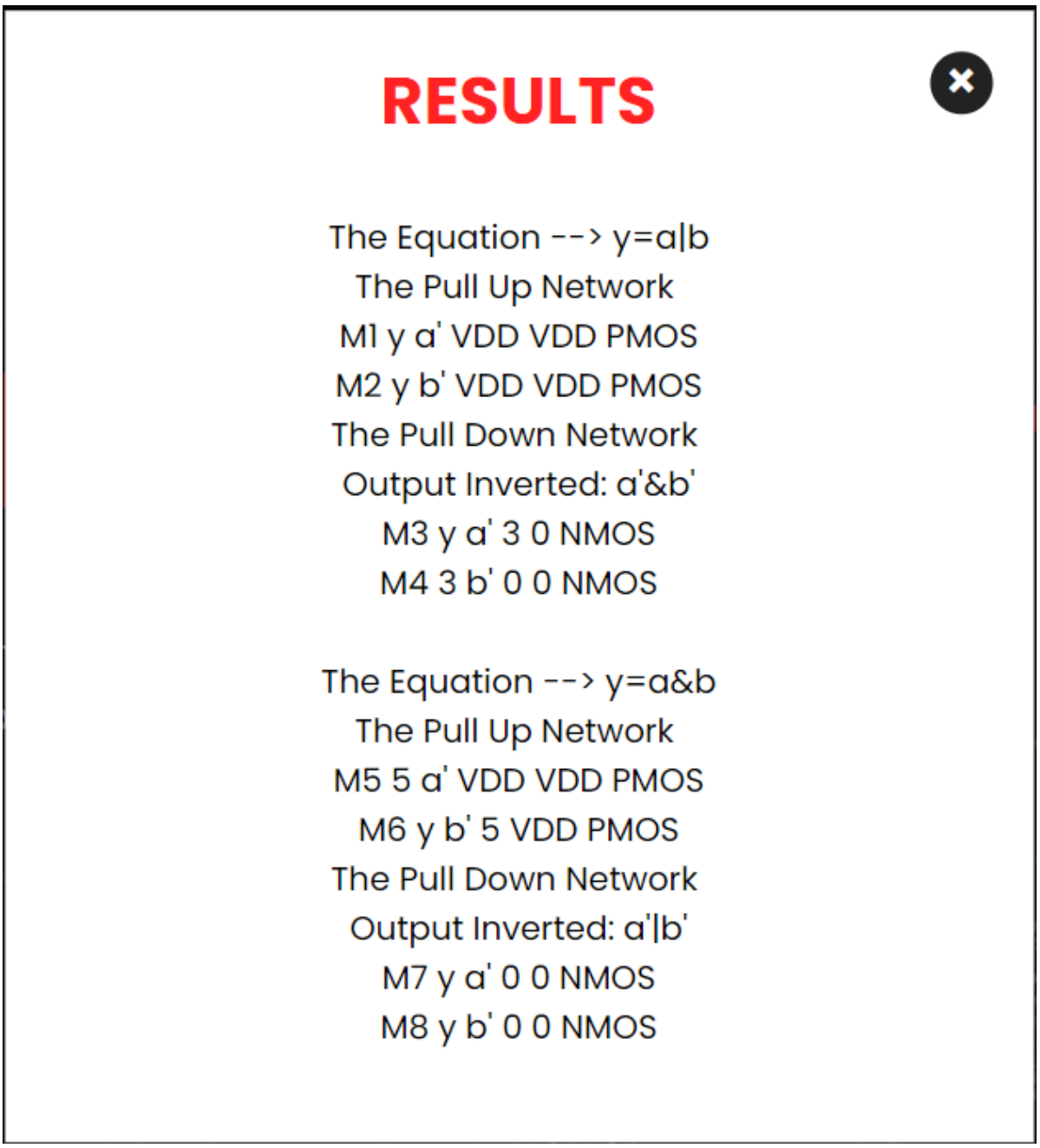
The multiple boolean functions must be separated with semi-colon so the input is separated into several CMOS that is being processed one at a time, after that. Then, the result is concatenated and sent into the front-end, web application, so it can be displayed for the user.

5

Test cases and Results

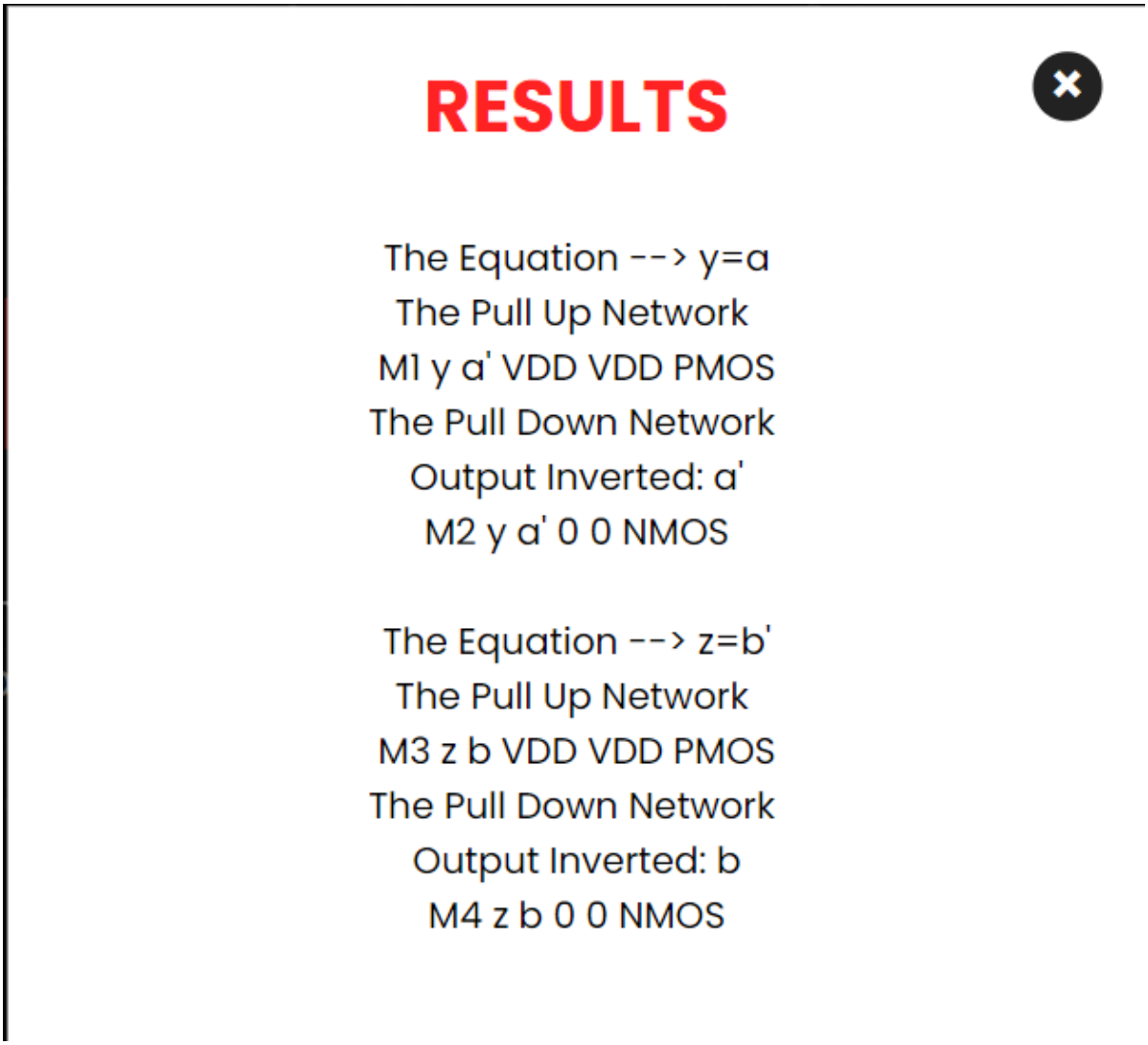
5.1. Test (1)



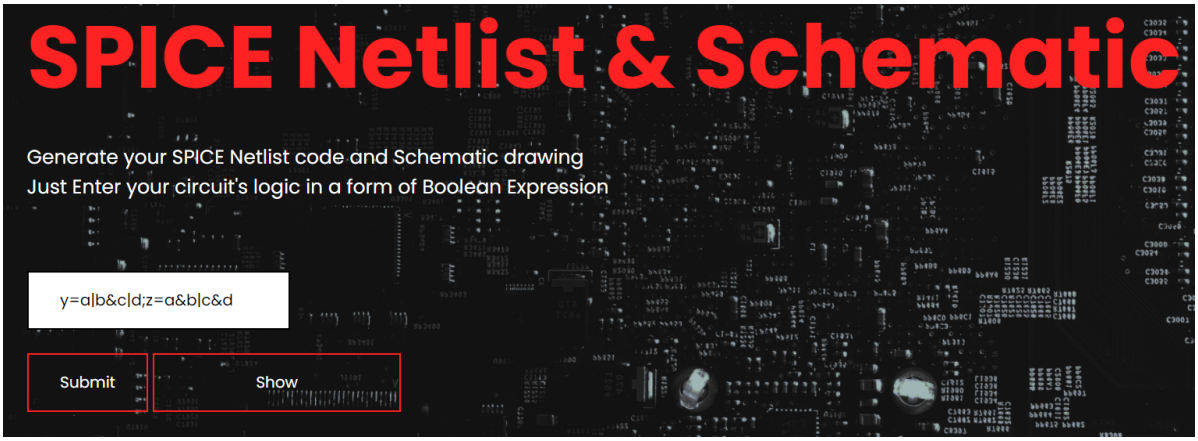

RESULTS
The Equation --> y=a|b
The Pull Up Network
M1 y a' VDD VDD PMOS
M2 y b' VDD VDD PMOS
The Pull Down Network
Output Inverted: a'&b'
M3 y a' 3 0 NMOS
M4 3 b' 0 0 NMOS
The Equation --> y=a&b
The Pull Up Network
M5 5 a' VDD VDD PMOS
M6 y b' 5 VDD PMOS
The Pull Down Network
Output Inverted: a'|b'
M7 y a' 0 0 NMOS
M8 y b' 0 0 NMOS

5.2. Test (2)


SPICE Netlist & Schematic
Generate your SPICE Netlist code and Schematic drawing
Just Enter your circuit's logic in a form of Boolean Expression
y=a;z=b'
Submit
Show


RESULTS
The Equation --> y=a
The Pull Up Network
M1 y a' VDD VDD PMOS
The Pull Down Network
Output Inverted: a'
M2 y a' 0 0 NMOS
The Equation --> z=b'
The Pull Up Network
M3 z b VDD VDD PMOS
The Pull Down Network
Output Inverted: b
M4 z b 0 0 NMOS

5.3. Test (3)

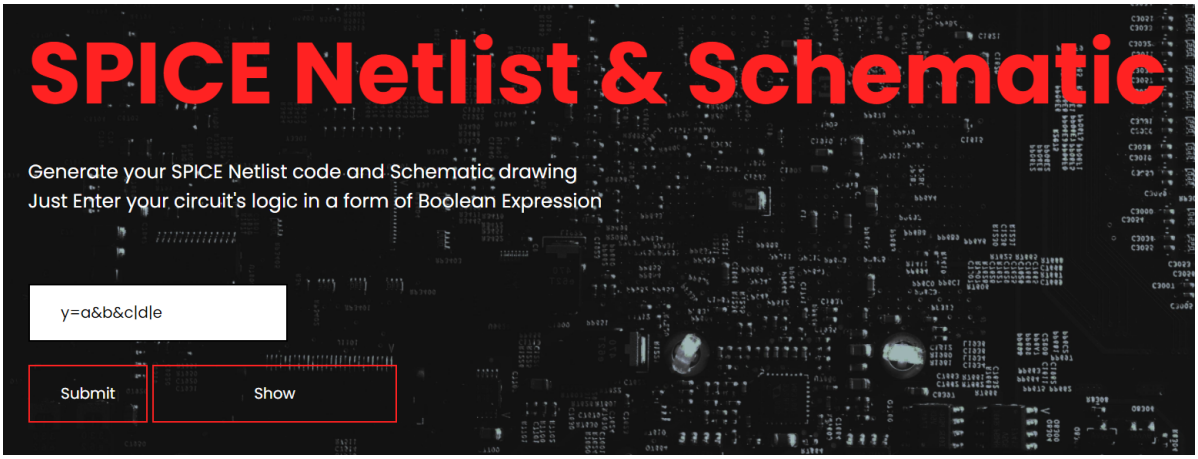

SPICE Netlist & Schematic
Generate your SPICE Netlist code and Schematic drawing
Just Enter your circuit's logic in a form of Boolean Expression
y=a|b&c|d;z=a&b|c&d
Submit
Show

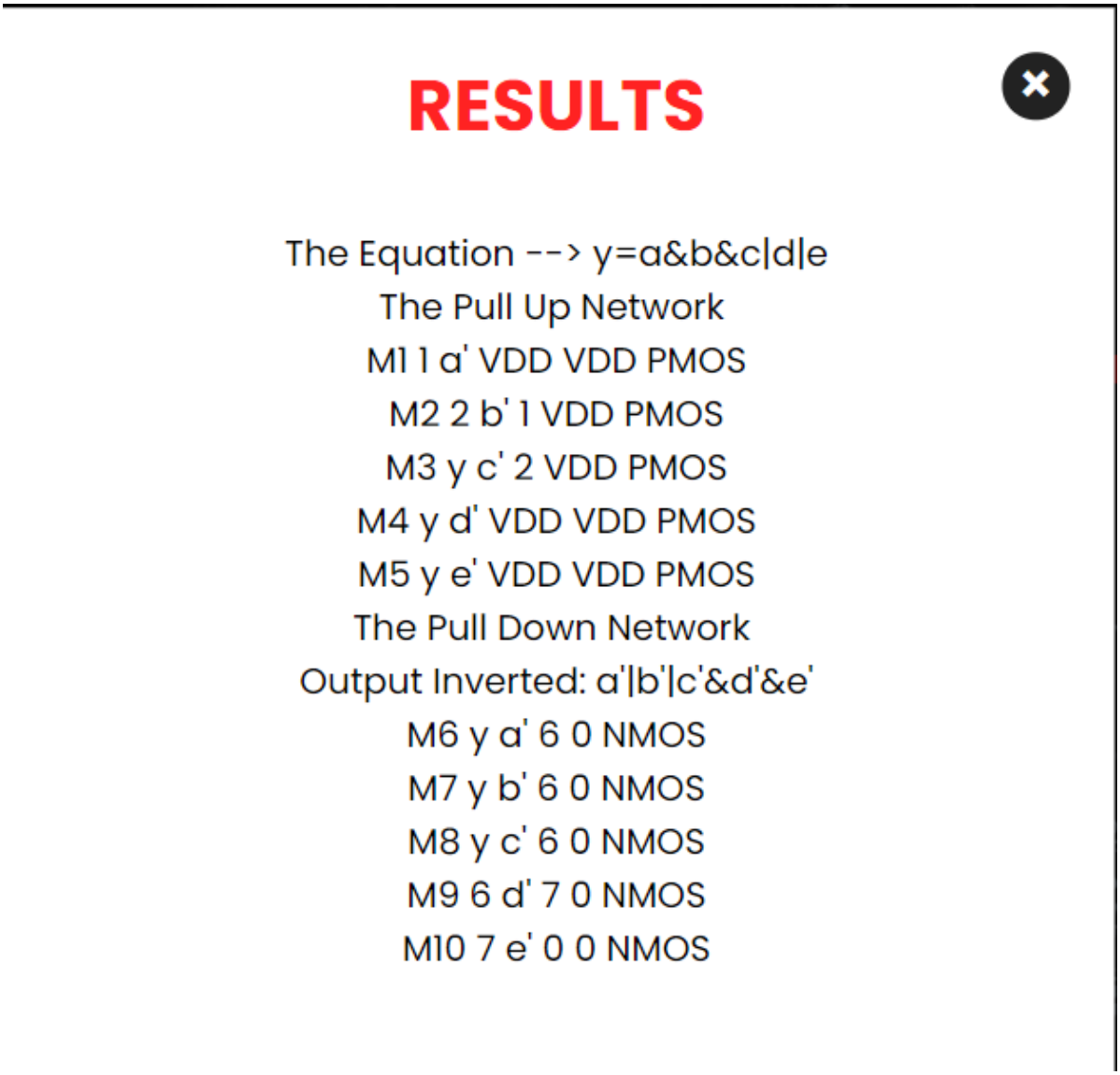
RESULTS

The Equation --> y=a|b&c|d
The Pull Up Network
M1 y a' VDD VDD PMOS
M2 2 b' VDD VDD PMOS
M3 y c' 2 VDD PMOS
M4 y d' VDD VDD PMOS
The Pull Down Network
Output Inverted: a'&b'|c'&d'
M5 y a' 5 0 NMOS
M6 5 b' 6 0 NMOS
M7 5 c' 6 0 NMOS
M8 6 d' 0 0 NMOS

The Equation --> z=a&b|c&d
The Pull Up Network
M9 8 a' VDD VDD PMOS
M10 z b' 8 VDD PMOS
M11 10 c' VDD VDD PMOS
M12 z d' 10 VDD PMOS
The Pull Down Network
Output Inverted: a'|b'&c'|d'
M13 z a' 12 0 NMOS
M14 z b' 12 0 NMOS
M15 12 c' 0 0 NMOS
M16 12 d' 0 0 NMOS

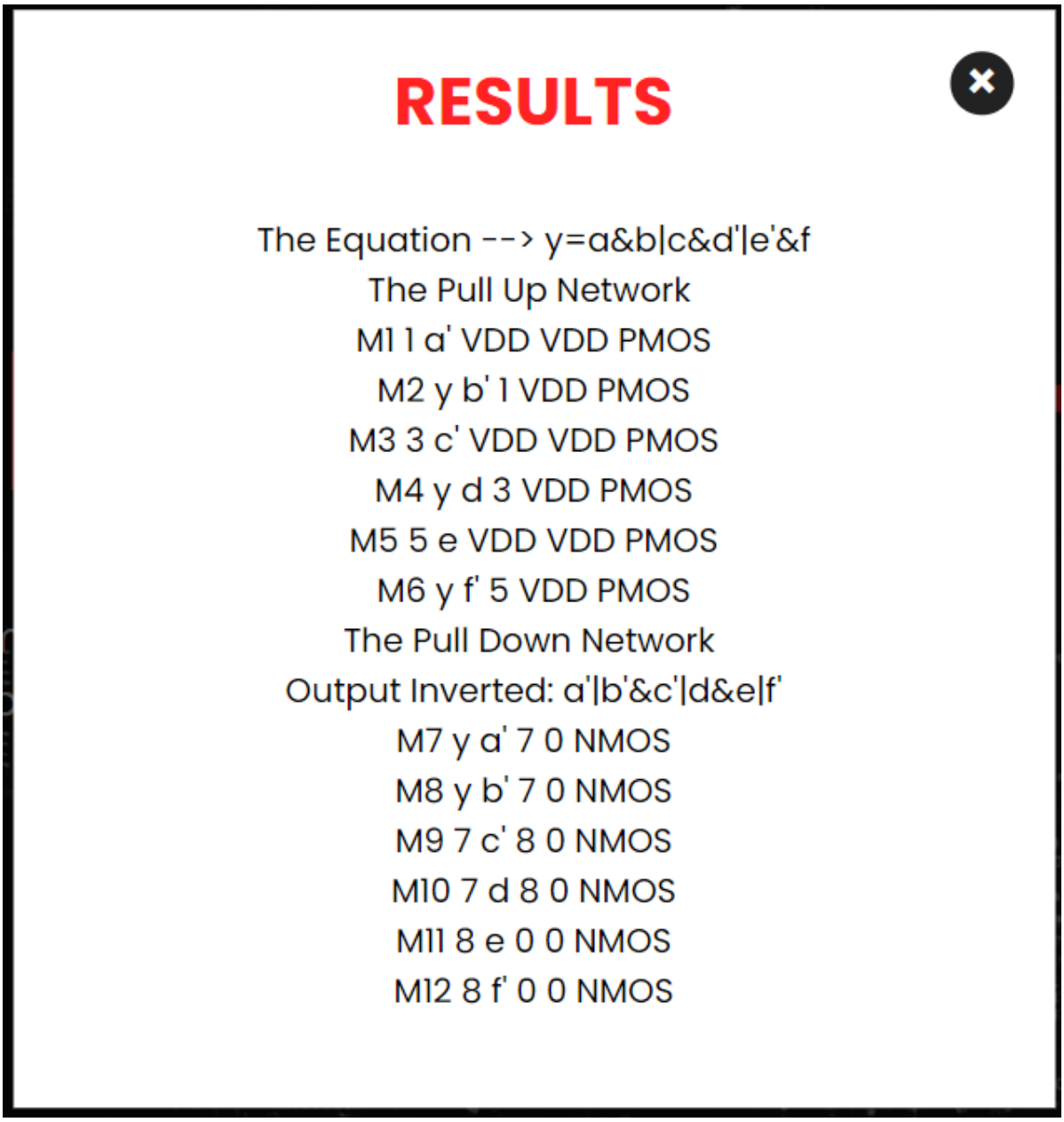
5.4. Test (4)


SPICE Netlist & Schematic
Generate your SPICE Netlist code and Schematic drawing
Just Enter your circuit's logic in a form of Boolean Expression
y=a&b&c|d|e
Submit
Show


RESULTS
The Equation --> y=a&b&c|d|e
The Pull Up Network
M1 1 a' VDD VDD PMOS
M2 2 b' 1 VDD PMOS
M3 y c' 2 VDD PMOS
M4 y d' VDD VDD PMOS
M5 y e' VDD VDD PMOS
The Pull Down Network
Output Inverted: a'|b'|c'&d'&e'
M6 y a' 6 0 NMOS
M7 y b' 6 0 NMOS
M8 y c' 6 0 NMOS
M9 6 d' 7 0 NMOS
M10 7 e' 0 0 NMOS

5.5. Test (5)


SPICE Netlist & Schematic
Generate your SPICE Netlist code and Schematic drawing
Just Enter your circuit's logic in a form of Boolean Expression
y=a&b|c&d'|e'&f
Submit
Show


RESULTS
The Equation --> y=a&b|c&d'|e'&f
The Pull Up Network
M1 1 a' VDD VDD PMOS
M2 y b' 1 VDD PMOS
M3 3 c' VDD VDD PMOS
M4 y d 3 VDD PMOS
M5 5 e VDD VDD PMOS
M6 y f' 5 VDD PMOS
The Pull Down Network
Output Inverted: a'|b'&c'|d&e|f'
M7 y a' 7 0 NMOS
M8 y b' 7 0 NMOS
M9 7 c' 8 0 NMOS
M10 7 d 8 0 NMOS
M11 8 e 0 0 NMOS
M12 8 f' 0 0 NMOS

A

Source Code

Listing A.1: ui.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" name="viewport" content="width=device-width">
    <title>SPICE Netlist & Schematic</title>
    <link rel="icon" type="image/icon type" href="https://i.ibb.co/FJ5ZZ4L/cpu-1.png" />
    <style>
        @import url('https://fonts.googleapis.com/css?family=Poppins:200
            ,300,400,500,600,700,800,900&display=swap');
        *
        {
          margin: 0;
          padding: 0;
          box-sizing: border-box;
          font-family: 'Poppins', sans-serif;
        }
        .button {
          color: white;
          padding: 16px 32px;
          font-size: 16px;
          transition-duration: 0.4s;
          cursor: pointer;
        }

        .upload {
          color: black;
          border: 2px solid #000000;
        }

        .upload:hover {
          background-color: #000000;
          color: white;
        }

        .predict {
          background: none;
          color: black;
          border: 2px solid #ff2222;
        }

        .predict:hover {
          background-color: #ff2222;
          color: white;
        }

        .image {
          width: 120px;
          height: 100px;
```

```
        padding-left: 50px;
        padding-top: 30px;
      }

      body {
        background-image: url('https://i.imgur.com/ObPJjEN.jpeg');
        background-repeat: no-repeat;
        background-attachment:fixed;
        background-size:cover;
        overflow: auto;
        min-height: 100%;
        min-width: 100%;
        position: absolute;
        box-shadow:inset 0 0 0 2000px rgba(153, 153, 153, 0.11);
      }
      .popup .overlay {
        position:absolute;
        top:0px;
        left:0px;
        bottom: 0px;
        right: 0px;
        min-height: 100%;
        min-width: 100%;
        background:rgba(0, 0, 0, 0.7);
        z-index:1;
        display:none;
      }

      .popup .content {
        position:absolute;
        top:50%;
        left:50%;
        transform:translate(-50%,-50%) scale(0);
        background:#fff;
        width:500px;
        height:auto;
        z-index:2;
        text-align:center;
        padding:20px;
        box-sizing:border-box;
        font-family:"Open Sans",sans-serif;
      }

      .popup .close-btn {
        cursor:pointer;
        position:absolute;
        right:20px;
        top:20px;
        width:30px;
        height:30px;
        background:#222;
        color:#fff;
        font-size:25px;
        font-weight:600;
        line-height:30px;
        text-align:center;
        border-radius:50%;
      }

      .popup.active .overlay {
        display:block;
      }

      .popup.active .content {
        transition:all 300ms ease-in-out;
        transform:translate(-50%,-50%) scale(1);
      }
    </style>
    <script>
    function togglePopup(){
      document.getElementById("popup-1").classList.toggle("active");
```

```
    }
    </script>
</head>
<body >
<img class = "image" src="https://i.ibb.co/McT96bK/cpu.png" alt="logo" border="0">
<h1 style="color:#ff2222 ; font-size: 90px; padding-left: 50px; padding-top: 200px;
    padding-right: 50px;" >SPICE Netlist & Schematic</h1>
<p style="padding-left: 50px; padding-top: 20px; font-size: 20px; color: white;
    padding-right: 50px;">
  Generate your SPICE Netlist code and Schematic drawing
  <br>
  Just Enter your circuit's logic in a form of Boolean Expression
</p>
<div id="wavContainer" style=" padding-bottom: 140px;">
    <form style="padding-right: 50px; padding-left: 50px; padding-top: 70px;" method="post"
    enctype="multipart/form-data">
        <input class="button upload" type="text" pattern="[a-zA-Z]{1}=[a-zA-Z&|';=]{1,}"
    title="Enter in the form of (x=y&z), use only alphabets and (&,|,')." name="equation"/>
        <br>
        <br>
        <input type="submit" style="color:#fff;" id="submitButton" class="button predict"
    value="Submit"/>
        <input readonly="readonly" style=" width: 255px; text-align: center; color:#fff;"
    onclick="togglePopup()" class="button predict" value="Show"/>
    </form>
</div>
<div class="popup" id="popup-1">
  <div class="overlay"></div>
  <div class="content">
    <div class="close-btn" onclick="togglePopup()">&times;</div>
    <h1 style="color:#ff2222">RESULTS</h1>
    <pre style="padding-top: 30px;">{{result}}</pre>
  </div>
</div>
</body>
</html>
```

Listing A.2: main.cpp

```
#include"crow_all.h"
#include <bits/stdc++.h>

using namespace std;

int k = 1;
int wire = 1;

void invertInputs(vector<vector<string>>&inputs)
{
    for(int i =0 ; i<(int)inputs.size(); i++)
    {
        for(int j = 0; j<(int)inputs[i].size(); j++)
        {
            if(inputs[i][j].size() < 2)
            {
                inputs[i][j].insert(inputs[i][j].begin()+1,'\'');
            }
            else
            {
                inputs[i][j].erase(inputs[i][j].begin()+1);
            }
        }
    }

}

struct CMOS {
    string name, drain, gate, source, body, type;
    CMOS();
    CMOS (string,string,string,string,string,string);
    void print_netlist();
```

```
33      string get_netlist();
34  };
35
36  CMOS::CMOS() {
37      this->name = ""; this->drain = ""; this->gate = ""; this->source = ""; this->body = "";
            this->type = "";
38  }
39
40  CMOS::CMOS(string name, string drain, string gate, string source, string body, string type )
        {
41      this->name = name; this->drain = drain; this->gate = gate; this->source = source; this->
            body = body; this->type = type;
42  }
43  void CMOS::print_netlist() {
44      cout << this->name << " " << this->drain << " " << this->gate << " " << this->source << "
             " << this->body << " " << this->type << "\n";
45  }
46  string CMOS::get_netlist() {
47      return this->name + " " + this->drain + " " + this->gate + " " + this->source + " " +
            this->body + " " + this->type + "\n";
48  }
49
50  string pullUp(string logicFunction, char in)
51  {
52      string netlist = "";
53      netlist += "The Pull Up Network \n";
54      vector<string> boolFunctions;
55      istringstream ss(logicFunction);
56      string str;
57      while(getline(ss, str, '|'))
58      {
59          boolFunctions.push_back(str);
60      }
61      vector<vector<string>> inputs((int)boolFunctions.size());
62      for(int i = 0; i<(int)boolFunctions.size(); i++)
63      {
64
65          istringstream splitByAnd(boolFunctions[i]);
66          string s;
67
68          while(getline(splitByAnd, s, '&'))
69          {
70              inputs[i].push_back(s);
71          }
72      }
73      invertInputs(inputs);
74      for(auto u : inputs)
75      {
76          for(auto k:u)
77          {
78              cout << k << endl;
79          }
80      }
81      vector<CMOS> vec;
82      for(int i = 0; i<(int)inputs.size(); i++)
83      {
84          for(int j = 0; j<(int)inputs[i].size(); j++)
85          {
86              string name = string(1,'M') + to_string(k);
87              CMOS c;
88              if(j==0 && (int)inputs[i].size() == 1)
89                  c = CMOS(name, string(1, in), inputs[i][j], "VDD", "VDD", "PMOS");
90              else if(j==0)
91                  c = CMOS(name, to_string(wire), inputs[i][j], "VDD", "VDD", "PMOS");
92              else if(j == (int)inputs[i].size()-1)
93                  c = CMOS(name, string(1, in), inputs[i][j], to_string(wire-1), "VDD", "PMOS")
                        ;
94              else
95                  c = CMOS(name, to_string(wire), inputs[i][j], to_string(wire-1), "VDD", "PMOS
                        ");
96              wire++;
```

```
            k++;
            vec.push_back(c);
        }
    }
    for(int i = 0; i<(int)vec.size(); i++)
    {
        cout << vec[i].get_netlist() << endl;
        netlist += vec[i].get_netlist();
    }
    return netlist;
}
string deMorgan(string boolFunction)
{
    for(int i = 0; i<(int)boolFunction.size(); i++)
    {
        if(boolFunction[i] == '&')
            boolFunction[i] = '|';
        else if(boolFunction[i] == '|')
            boolFunction[i] = '&';
        else if( (boolFunction[i]>='a' && boolFunction[i]<='z' && boolFunction[i+1] !='\'')
            || (boolFunction[i]>='A' && boolFunction[i]<='Z' && boolFunction[i+1] !='\'') )
            boolFunction.insert(boolFunction.begin()+i+1,'\'');
        else if( (boolFunction[i]>='a' && boolFunction[i]<='z' && boolFunction[i+1] =='\'')
            || (boolFunction[i]>='A' && boolFunction[i]<='Z' && boolFunction[i+1] == '\'') )
            boolFunction.erase(boolFunction.begin()+i+1);
    }
    return boolFunction;
}

string pullDown(string logicFunction, char in)
{
    string netlist = "";
    netlist += "The Pull Down Network \n";
    string outputInverted = deMorgan(logicFunction);
    netlist += "Output Inverted: " + outputInverted + "\n";
    vector<string> boolFunctions;
    istringstream ss(outputInverted);
    string str;
    while(getline(ss, str, '&'))
    {
        boolFunctions.push_back(str);
    }
    vector<vector<string>> inputs((int)boolFunctions.size());
    for(int i = 0; i<(int)boolFunctions.size(); i++)
    {

        istringstream splitByOR(boolFunctions[i]);
        string s;

        while(getline(splitByOR, s, '|'))
        {
            inputs[i].push_back(s);
        }
    }
    vector<CMOS> vec;
    for(int i = 0; i<(int)inputs.size(); i++)
    {
        for(int j = 0; j<(int)inputs[i].size(); j++)
        {
            string name = string(1,'M') + to_string(k);
            CMOS c;
            if(i==0 && (int)inputs.size() == 1)
                c = CMOS(name, string(1,in), inputs[i][j], string(1,'0'), string(1,'0'), "
                    NMOS");
            else if(i==0)
                c = CMOS(name, string(1,in), inputs[i][j], to_string(wire), string(1,'0'), "
                    NMOS");
            else if(i == (int)inputs.size()-1)
                c = CMOS(name, to_string(wire-1), inputs[i][j], string(1,'0'), string(1,'0'),
                     "NMOS");
            else
```

```
                c = CMOS(name, to_string(wire-1), inputs[i][j], to_string(wire), string(1,'0'
                    ), "NMOS");
            k++;
            vec.push_back(c);
        }
        wire++;
    }
    for(int i = 0; i<(int)vec.size(); i++)
        netlist += vec[i].get_netlist();

    return netlist;
}

bool valid_input(string eq)
{
    int index = eq.substr(1,eq.size()-1).find(eq[0]);
    if(index < 0 || index >= eq.size()){
        return true;
    }
    return false;
}

void semi(string &s)
{
    if(s[s.size()-2] != ';')
    {
        s[s.size()-1] = ';';
        cout << s << endl;
    }
}

string get_result(string input)
{
    if(!valid_input(input))
    {
        return "The Input is not Valid.\nOutput Must be Different from Inputs.\n";
    }
    string output;
    string result = "The Equation --> ";
    if(input[1] != '\'')
        output = input.substr(2,input.size()-2);
    else
        output = input.substr(3,input.size()-2);
    result += input + "\n";
    result += pullUp(output, input[0]);
    result += pullDown(output, input[0]);
    return result;
}

string solve(string eqs)
{
    semi(eqs);
    cout << eqs << endl;
    istringstream s(eqs);
    string eq;
    vector<string> equations;
    string results = "";
    while(getline(s, eq, ';'))
    {
        if(eq.size() > 1)
        {
            cout << eq << endl;
            equations.push_back(eq);
        }
    }
    for(auto u: equations)
    {
        results += get_result(u) + "\n";
    }
    return results;
}
```

```
233
234 int main()
235 {
236     crow::SimpleApp app;
237     auto page = crow::mustache::load("ui.html");
238
239     CROW_ROUTE(app, "/").methods(crow::HTTPMethod::GET, crow::HTTPMethod::POST)([page](const
        crow::request& req){
240         if (req.method == crow::HTTPMethod::POST)
241         {
242             stringstream ss(req.body);
243             string eq; vector<string> lines;
244             while(getline(ss, eq))
245             {
246                 lines.push_back(eq);
247             }
248             eq = lines[3];
249             string result = solve(eq);
250             crow::json::wvalue wv = {{"result", result}};
251             crow::mustache::context &ctx = wv;
252             cout << "POST" << endl;
253             k = 1;
254             wire = 1;
255             return page.render(ctx);
256         }
257         else
258         {
259             cout << "GET" << endl;
260             return page.render();
261         }
262     });
263
264     //set the port, set the app to run on multiple threads, and run the app
265     app.port(18080).multithreaded().run();
266 }
```

B

Task Division

Table B.1: Distribution of the workload

Task	Student Name(s)
OOP	Yehia & Rehab
Multiple input	Gohar
DeMorgan	Rehab & Yehia
Pull Up Network	Rehab
Pull Down Network	Yehia
Website Front-end	Gohar
Website Back-end	Gohar
Testing and Validation	Rehab & Gohar & Yehia
Report Writing	Gohar & Rehab & Yehia
Report Design	Gohar & Rehab & Yehia