Technische Universität München
Department of Informatics
Department of Electrical Engineering and Information Technology
Institute of Electronic Design Automation

Integration of Columba – Design Automation for Microfluidics – as a Web Service with Web User Interface

Bachelor's Thesis in Informatics

Yushen Zhang

Technische Universität München
Department of Informatics
Department of Electrical Engineering and Information Technology
Institute of Electronic Design Automation

Integration von Columba

– Automatisierter Entwurf für Mikrofludik –

als ein Webservice mit Web-Oberfläche

Integration of Columba

– Design Automation for Microfluidics –

as a Web Service with Web User Interface

Bachelor's Thesis in Informatics

Yushen Zhang

Advisor:	Dr.-Ing. Tsun-Ming Tseng
Supervisor:	Prof. Dr. sc.techn. Andreas Herkersdorf
Submission Date:	15.09.2017

Confirmation

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Date: 15.09.2017 Signed: Yushen Zhang

Abstract

Columba is a design automation tool for continuous-flow microfluidic large-scale integration developed by Dr.-Ing. Tsun-Ming Tseng. It is the first design automation tool that can seamlessly synchronize with the manufacturing flow (Tseng, et al., 2017). Columba was originally developed for Linux system. On the one hand, it can only be used by users with a Linux environment. On the other hand, it demands users to have concrete Linux knowledge to install pendency programs. In the hope that users, who are using another operating system or are not familiar with Linux system, can design their chips as well, we provide a web service for our users to access Columba via a web interface so that the users can design their chips simply on a web browser within any device.

Abstrakt

Columba ist ein Design-Automatisierungswerkzeug für die kontinuierliche mikrofluidische Großintegration entwickelt von Dr.-Ing. Tsun-Ming Tseng. Es ist das erste Design-Automatisierungswerkzeug, das sich nahtlos mit dem Fertigungsablauf synchronisieren kann (Tseng, et al., 2017). Columba wurde ursprünglich für Linux-System entwickelt. Einerseits kann es nur von Benutzern mit einer Linux-Umgebung genutzt werden. Andererseits fordert es die Benutzer konkretes Linux-Wissen zu haben um vorausgesetzte Softwareumgebung breitzustellen. In der Hoffnung, dass Benutzer, die ein anderes Betriebssystem verwenden oder nicht mit Linux-System vertraut sind, auch ihren Chip entwerfen können, bieten wir also einen Web-Service für unsere Benutzer. Damit können die Benutzer über ein Web-Interface auf Columba zugreifen, und ihre Chip-Entwürfe einfach mit einem Webbrowser auf jedem Gerät entwerfen.

Acknowledgements

I would like to give all of them, who supported me during this work, my special thanks. Especially, I would like to give my advisor Dr.-Ing. Tsun-Ming Tseng and Prof. Dr.-Ing. Ulf Schlichtmann warm thanks for providing me this exciting and expressive topic as my bachelor's thesis. With their support and help I was able to finish this thesis without hassle. I am deeply convinced that all the practices I have had during my work will help me in the future, whether it is in my academic life or in my career. Finally, I would like to thank Prof. Dr. sc.techn. Andreas Herkersdorf very much for being my supervisor. Without him, I would not be able to do this fascinating topic at the Institute of Electronic Design Automation.

Contents

List of Figures

List of Table

List of Codes

1. Introduction

With the increase of complexity of microfluidic applications, the design process of complicated layer becomes difficult. Manual design of such a complex design turns out to be time-consuming and error-prone. Automated design of microfluidics emerged. Columba 2.0, developed by Dr.-Ing. Tsun-Ming Tseng, generates AutoCAD-compatible designs that fulfill all design rules and can be used directly for mask fabrication. Columba takes plain-text netlist descriptions as inputs, and performs simultaneous placement and routing for multiple layers while ensuring the planarity of each layer. (Tseng, et al., 2017) Thus, by using Columba, microfluidic designs can be generated automatically, which greatly saves time.

The original Columba was implemented as a Linux program and requires user to install Gurobi Optimizer (Gurobi Optimization, Inc., 2017) to work. The install process of Gurobi Optimizer is complicated. The users need to know how to setup the required system environment path and retrieve a license by using Linux terminal. Seeing that most of the biologists and chemists may not possess much knowledge about Linux and in particular do not have the required knowledge for installing Gurobi Optimizer, it can be very cumbersome to use the original Columba. And Columba only runs on Linux, which makes it more difficult for other operating system users to use, since they must switch the operating system.

A way to solve this problem might be to implement Columba for different operating system. But still, this would require installation of dependent software and libraries. Consequently, to ease the process of using Columba, the idea of having an online web Columba application has emerged. In contrast to implementing Columba for different operating system, a web-based application requires nothing but a web browser to open the website. More importantly, web-based Columba is platform-independent. The users can design a chip on any kind of device instantly, whether it is a smartphone or a personal computer.

This bachelor's thesis develops the idea of building a web service platform, on which scientists can generate their microfluidic designs with ease. This whole work implements the integration of Columba as a web service with a web user interface – Cloud Columba.

This bachelor's thesis is organized as the follows: Chapter 2 will give the background and a concise introduction to Columba and its role within this system. A description of task, requirements and expectations of this work will be described in chapter 3. Afterwards, in chapter 4, concrete software design and concept of this system will be described. And then, the way in which it was implemented and problems that have occurred during implementation and their solutions will be stated in chapter 5. Lastly, an outlook and summary will be made in chapter 6.

2. Background

2.1. Microfluidic Biochip Design

The flow-based microfluidic biochips are currently being designed manually using the drawing computer-aided design (CAD) programs, e.g., AutoCAD (Stanford Microfluidic Foundry, 2017). A complete understanding of the application is required for the designer to design a microfluidic chip that fulfills the requirements. At the same time, the designer also needs to have complete knowledge and skills to ensure that all design rules, for instance, channel thickness and height, height-to-width aspect ratio, and spacing between channels and the punch holes, are satisfied. (Hu, et al., 2017)

2.2. Design Automation

As the microfluidic applications become more and more popular and more complex, and with the increase in the number of on-chip valves and valve densities, the need of automated designs for microfluidics becomes necessary. Manually designing a complex constructed microfluidic biochip is very unpractical. On the one hand, it is time-consuming and on the other hand, it is a complex and error-prone process. Thus, design automation for microfluidic biochip appeared. Previous design automation approaches used to design each microfluidic layer separately and over-simplify the layer interactions to various degrees, which resulted in a gap between realistic requirements and automatically-generated designs. (Tseng, et al., 2017)

2.3. Columba 2.0

Columba, as a microfluidic large-scale integration design automation tool, generates AutoCAD-compatible designs that fulfill all designs rules and can be directly used for mask fabrication, was developed.

In this work, we used Columba ver. 2.0 for microfluidic biochip design generation, and integrate its core algorithm into our web service.

Columba ver. 2.0 uses four types of module models: mixer, reaction chamber switch and inlet/outlet. Columba models the physical synthesis problem as a linear optimization problem, and uses Gurobi Optimizer to solve the problem progressively in four sequential phases: global layout generation, pin allocation, inlet/outlet restoration and refinement. Figure 1 shows how the complete chip production process is supported by Columba.

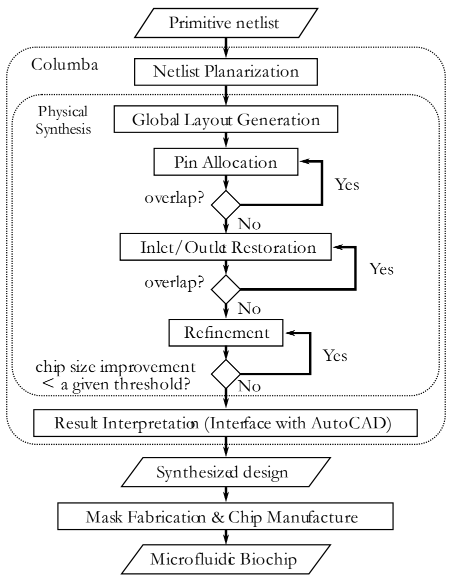


Figure 1: Chip production process of Columba (Tseng, et al., 2017)

On the program side, Columba ver. 2.0 takes a design rule file, a design netlist description and four parameters as input. Its output is an AutoCAD script of the design, which can be directly used for mask fabrication and chip manufacture.

Columba ver. 2.0 is written in C++ and was built to run on Linux. It requires Gurobi Optimizer ver. 7.0 pre-installed on the system and needs to be configured correctly, especially set in system environment path. The Gurobi Optimizer makes use of several executable. In order to allow these to be found when needed, modification of a few environment path variables is required. To use Gurobi Optimizer, a license must be retrieved and registered, otherwise Columba cannot invoke it and cannot work as expected.

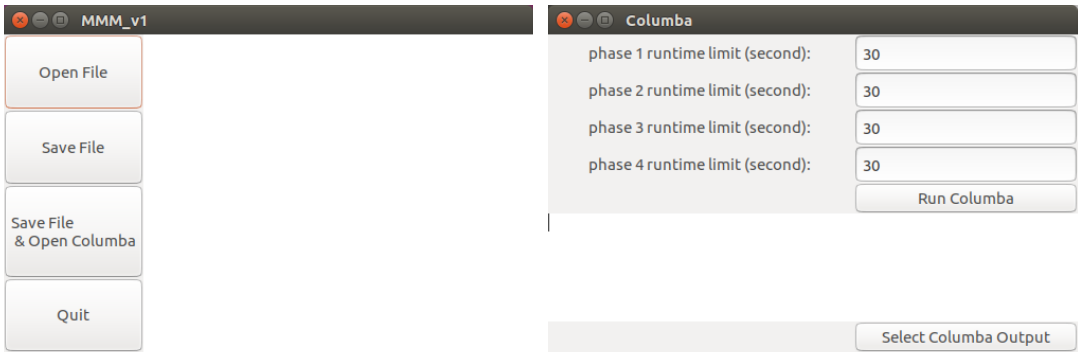
Currently, there is no approach integrating a design automation tool as a web-based platform. Thus, this work is the first web-based design automation tool for continuous-flow microfluidic large-scale integration.

3. Task and Requirements

The idea of building a web-based microfluidics automated design generation platform using Columba originally evolved from the idea of implementing Columba for different operating systems, such as Microsoft Windows and Apple macOS. We had the idea to implement a cross-platform application, such as a Java applet, which is capable to run on multiple operating system without rebuilding it. But that turned out to be inappropriate as well. The user still needed to configure Gurobi Optimizer, which might be difficult for normal user.

Under those circumstances, we integrated Columba into a web service platform. Accordingly, users no longer need to concern about installing Columba and its required software, and can use this platform on any device at any time instantly, even on mobile devices, such as smartphones or tablets. The integrated web service platform has a web user interface, on which the following functions are available.

The functions available in the web service are the same as in the Columba program running on Linux: a rule and a script editor with load and save functions, input fields for parameter setting and a button to execute the generation process.



phase 1 runtime limit (second):	30
phase 2 runtime limit (second):	30
phase 3 runtime limit (second):	30
phase 4 runtime limit (second):	30

Figure 2: Columba 2.0 running under Linux (Ubuntu 17)

Moreover, we have decided to add a preview function. After the design has been generated as AutoCAD script, users will have a preview image displayed on the screen to view and check before importing the script into AutoCAD. As a web service, it is important to have the capability to generate multiple designs by multiple web browsers/users simultaneously. Therefore, multiple and parallel executions of Columba must be ensured.

In short, the platform provides an interactive user interface which allows users to open, edit and save their design rule and script. After inputting required parameters, an AutoCAD script will be generated along with an image preview of the design and be available for download on the website.

4. Concept and Design

4.1. System Concept

The basic concept of this integrated web system is depicted in Figure 3. As the figure shows, all users will communicate with the server via the world wide web. Users use the UI displayed on their browser and transfer information through the web to the server. The server runs Columba with the information given and returns finished design script and preview image back to the browser.

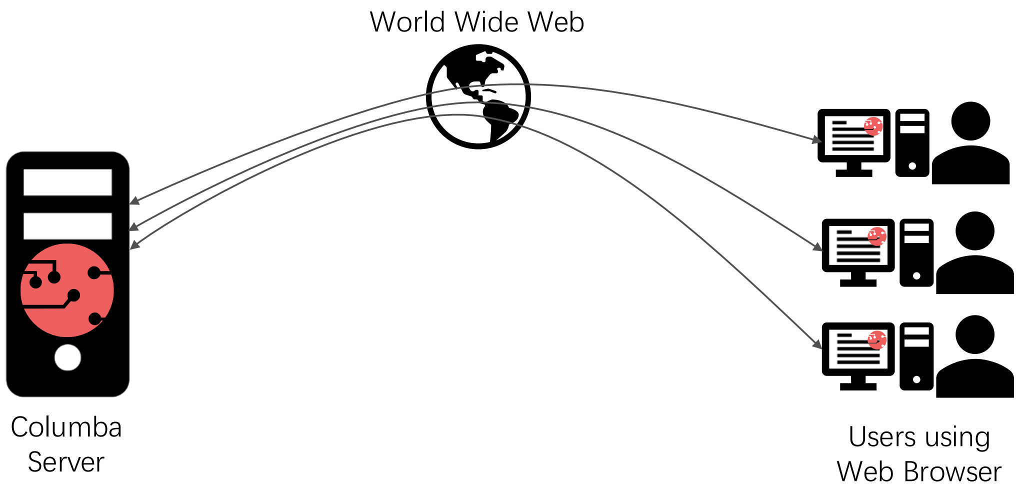


Figure 3: Basic concept of web-based Columba (Cloud Columba)

In general, a user opens Cloud Columba in her/his web browser and choose to start a new design. The web user interface provides several options. Users can open or implement their design rule and script and enter necessary parameters. After they click on "Generate" button, these inputs will be sent to the sever via HTTP-post method. Browser waits until server responds with the AutoCAD script and the preview image, and displays the image on the screen.

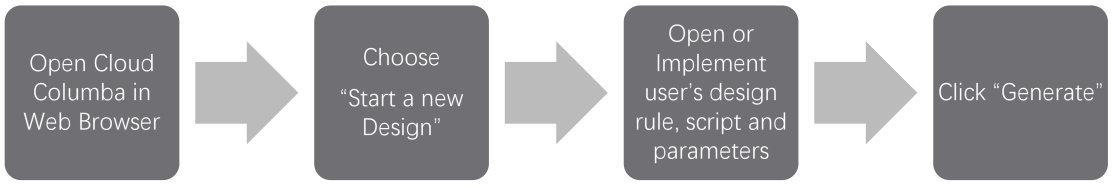

Open Cloud Columba in Web Browser
Choose "Start a new Design"
Open or Implement user's design rule, script and parameters
Click "Generate"

Figure 4: User action process

On the other site, the server waits until it receives the design rule, script and parameters from the user and then generates an AutoCAD script by invoking Columba core algorithm with the received information and a preview image. After this, it sends generated AutoCAD script and image back to the user.

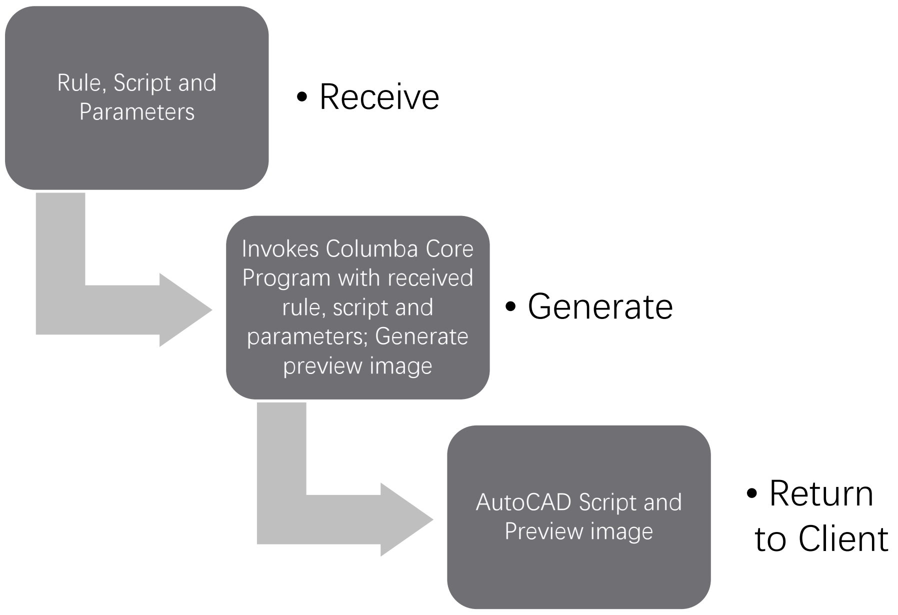

Rule, Script and Parameters
• Receive
Invokes Columba Core Program with received rule, script and parameters; Generate preview image
• Generate
AutoCAD Script and Preview image
• Return to Client

Figure 5: Server action process

4.2. System Design

According to our concept, we designed the system mainly as two components: front-end and back-end. Front-end is the user interface towards users, while back-end is the part which processes all information received from each user and controls the entire system. The back-end can be seen again as two parts: main controller and Columba core algorithm.

The front-end (user interface) is implemented with modern HTML5 including the use of CSS3 for content style and JavaScript for controls. On the back-end, we use PHP to implement our main controller and C++ for Columba core algorithm.

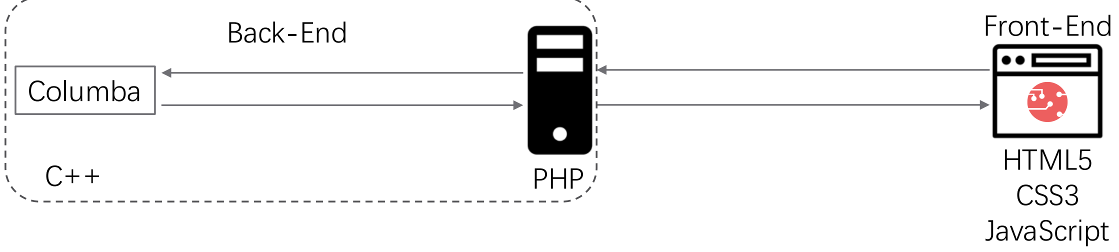


Figure 6: System Design

The main task of front-end is to provide a user interface which shows content and information sent by the server, such as input fields and finished design. Furthermore, it allows users to interact with the system, such as to open design rule and input parameters. The main controller in the middle, which is the main part of the back-end, forwards data both from the front-end and from Columba. Data received from the front-end will be used to execute Columba core algorithm to generate the design, and the finished design will be transformed and sent back to the front-end. Columba, as another part of the back-end, will do the data processing and calculation. Subsequently, we have applied the Model-View-Presenter design pattern to our system as it is depicted in Figure 7 (Potel, 1996).

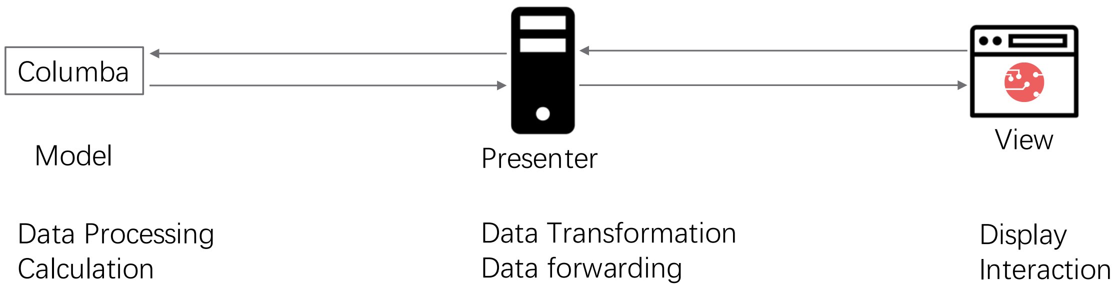


Figure 7: Model-View-Presenter applied on Cloud Columba

4.3. Server Environment

Although most of the web servers use Linux (W3Techs, 2017), we still decided to use Windows Server 2016 as the server's operating system. The main reason for this is its flexibility and compatibility with Windows applications. With regards to extendibility, we might extend our system in several ways in the future, and therefore we consider that Windows Server provides the most suitable environment.

For our server environment, we use Windows Server 2016 built-in Microsoft Internet Information Service (IIS) as our web server, installed along with PHP ver. 7.1 and Gurobi Optimizer ver. 7.0 required for Columba.

For this work, we use a server with following specification for testing purpose:

CPU	Intel Core i5-3427U 1.8Ghz-2.3Ghz
RAM	Kingston 2×8GB DDR3 1600Mhz
HDD	Crucial M550 512GB SSD
OS	Windows Server 2016 64-bit
Web Server	IIS ver. 10.0 + PHP ver. 7.1 64-bit

Table 1: Test server specification

5. Implementation

5.1. Front-End (View)

Though web browsers can display web pages, different devices provide different environment. For instance, mobile devices usually have a small touch screen instead of mouse cursor and a large monitor. To ensure our requirements that Cloud Columba is cross-platform and usable on any devices which have web-browser, we use the concept of Responsive Web Design.

Responsive Web Design is a web development approach that creates dynamic changes to the appearance of a website, depending on the screen size and orientation of the device being used to view it. In addition, it is important to understand that Responsive Web Design tasks include offering the same support to a variety of devices for a single website (Schade, 2014).

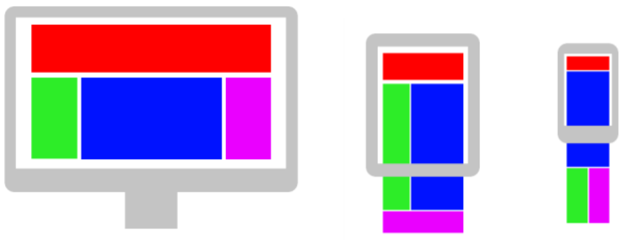


Figure 8: An example of how various elements of a web page adapt to the screen size of different devices (Wikipedia, 2017)

For our front-end UI design, we use a Responsive Web Design templet Material Dashboard, made by Creative Tim, with a fresh, modern design inspired by Google's Material Design (Tim, 2017). Furthermore, our front-end is crafted with Bootstrap – the most popular front-end framework (Bootstrap, 2017), and JQuery - a fast, small, and feature-rich JavaScript library (JQuery, 2017).

To start with, we use grid designs to locate different contents on separate places with defined width to setup our layout. Bootstrap framework provides full support for Responsive Web Design. To use its default grid system, we must set the main container with class attribute "container-fluid". We use “col-md-4” and “col-md-8” classes to setup columns. The first one is used for general information input, the second one for script input.

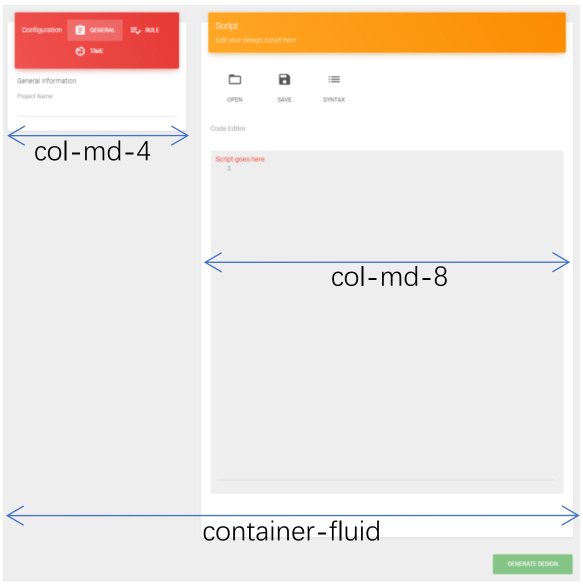


Figure 9: An example of Cloud Columba layout and style class name

Because of the support of Bootstrap framework, the layout will automatically be fitted for different screen sizes.

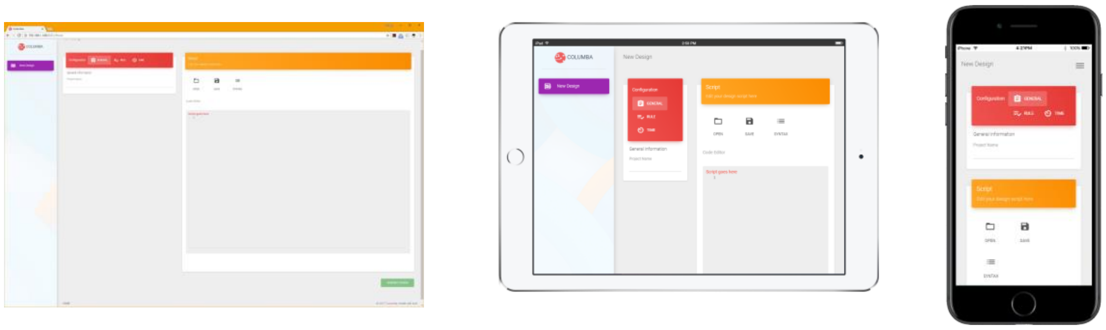


Figure 10: Columba's layout will automatically be fitted for different devices

Inputs are realized with standard input with type of text and number, and text area for rule and script editing. We use Bootstrap's integrated tab to divide inputs into three tabs and all of them are placed in the general information column. Open and save buttons are also realized as tabs with a picture, which will execute a defined JavaScript function. Finally, we have a "Generate" button for sending information to the server, which is realized using a Bootstrap button.

After we have setup the basic layout, we need to implement functions which are bound to certain actions. These functions are implemented with JavaScript using JQuery library. In the following we will explain how basic functions of Columba are realized in the front-end by using some code examples.

The open function is one of Columba's basic functions. We use an HTML input type file to open a file dialog and retrieve a file from the client.

```
<input type="file" id="openScript" style="display:none" name="script" accept=".clbs,.txt,text/txt,txt/clbs">
```

Code 1: File dialog

Next, we use a small JavaScript function to read the opened file and put the content into a given text area.

```
function readScriptFile(e) {
    var file = e.target.files[0];
    if (!file) {
        return;
    }
    var reader = new FileReader();
    reader.onload = function (e) {
        var contents = e.target.result;
        displayScript(contents);
    };
    reader.readAsText(file);
}
function displayScript(contents) {
    var element =
document.getElementById('script');
    element.value = contents;
}
```

Code 2: Read file and load it into text area

The save function reads the text from the specified text area, and converts it into a blob-file indicated with type plain-text and lets browsers to save it.

```
function saveText(sourceID, ext) {
    var script = $("#" + sourceID).val();
    var blob = new Blob([script], {type: "text/plain;charset=utf-8"});
    saveAs(blob, sourceID + "." + ext);
}
```

Code 3: Save content of text area

To avoid sending empty script, rule and parameters to the server, a check before sending is implemented. This uses Bootstrap's built-in function and works by marking inputs with the "required" tag.

```
<textarea class="form-control" rows="34" id="rules" required>
```

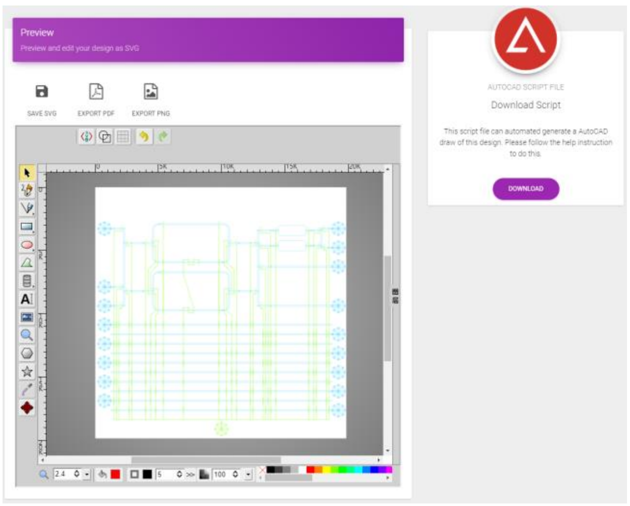
Code 4: Required input marked with "required"

After all inputs have been done, the user clicks generate to send them to the server. This button invokes an AJAX function which uses HTTP-post method to send.

```
$.post("newGenerate.php",
        {
            name: $("#projectName").val(),
            p1Time: $("#p1Time").val(),
            p2Time: $("#p2Time").val(),
            p3Time: $("#p3Time").val(),
            p4: $("#p4").val(),
            rules: $("#rules").val(),
            script: $("#script").val()
        });
```

Code 5: Send data to server using AJAX

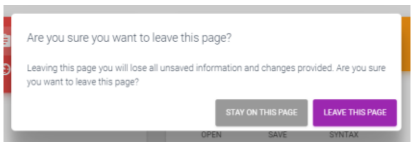
It waits until the server responds and changes the page to a result page, which has a preview section and an AutoCAD script download section. The layout is also realized with Bootstrap's grid system.



This script file can automated generate a AutoCAD draw of this design. Please follow the help instruction to do this.

Figure 11: Result page of Cloud Columba's generation

The preview function is powered by a JavaScript application SVG-Edit (Syazwan, et al., 2017). Apart from that, we also added dialog for warnings by using Bootstrap's modal which will be triggered by certain actions, for instance changing to another page before submitted. A JavaScript function will check if such an action is happening.



Are you sure you want to leave this page?

Leaving this page you will lose all unsaved information and changes provided. Are you sure you want to leave this page?

Figure 12: An example of warning dialog

5.2. Back-End

The Back-end running on the server has, as mentioned, two parts. The main controller, written in PHP script, answers requests sent from the client/front-end. On the one hand, clients will request the front-end web site (UI); on the other hand, clients will send data and request for design generation. To generate a design, the PHP script will execute Columba core algorithm, which is the other part of the back-end, with received data. In the following, the implementation method of the back-end will be descripted.

5.2.1. Main Controller (Presenter)

The main controller does mainly the following: get posted data from client/front-end, save script and rule as text files, execute Columba with these data, call the preview image conversion functions generate preview images, and send them to the front-end.

Therefore, the functions provided by the main controller in principal can be classified into two categories: execute Columba to generate the design and convert AutoCAD script into a preview image.

For our preview image file format, we use an XLM-based vector image file type – SVG (Scalable Vector Graphics). SVG is nowadays a widely used format and is recommended by World Wide Web Consortium (W3C) to describe vector graphics.

After the server receives a request for design generation, the server will save the script and rule as local files which will be used for executing Columba. To save files, we use PHP's fwrite function, and give the files a unique name, to prevent overwriting happening and to ensure that multiple executions are possible.

```
$scriptFile = fopen($INPUT_PATH . $name . "_script.txt", "w") or die("Unable to open file!");
  $ret = fwrite($scriptFile, $script);
  fclose($scriptFile);
```

Code 6: Store received script on server

Eventually, we execute Columba with defined parameters with PHP's exec function. In the next section, we will describe Columba which runs on the server with more details.

```
exec_cmd = $COLUMBA_EXE . " " . escapeshellarg($INPUT_PATH . $name . "_rule.txt") . " " .
escapeshellarg($INPUT_PATH . $name . "_script.txt") . " " . $p1Time . " " . $p2Time . " " . $p3Time . " " . $p4 . "
" . escapeshellarg($name) ;

exec($exec_cmd, $o, $ec);
```

Code 7: Execute Columba on server vie PHP exec function

After Columba has generated a design as an AutoCAD script, the file path will be sent to the front-end, and then another function will be executed converting the AutoCAD script to an SVG drawing. This function has many helper functions. To convert the AutoCAD script into an SVG drawing, we have to match all AutoCAD commands used in the script.

The script includes the following commands: layer, color, pline, arc direction, line, union and region. We use if expressions to match these commands, and try to do the same thing they are doing within AutoCAD on SVG.

The layer command creates a layer in AutoCAD, so we create a group in SVG which is the counterpart of layer in SVG. Color command uses AutoCAD Color Index(ACI) as parameter and sets the color of a layer. We defined a function to match all ACI colors into RGB values (Moses, 2017). This function is based on an array stored with RGB values, while the index of it is the ACI number. The remaining commands are drawing commands. All these commands can be directly translated into SVG descriptions, except arc direction, region and union, because of the same types of arguments they are using.

The arc direction command draws an arc according to the given parameters: a start point, an end point and a start direction tangent vector. In SVG description, it uses a start point, an end point, X-axis rotation, sweep-flag and radius to descript how the arc looks like. For that reason, we implemented a function to convert the parameters.

Unfortunately, for region and union commands, SVG does not provide similar options. Thus, we ignore these commands, which results overlapping shapes that are not unified. However, since the SVG drawing is only a design preview, these overlapping shapes do no harm to the AutoCAD designs.

Finally, the SVG preview image will be returned and sent to the front-end along with the AutoCAD script.

5.2.2. Modified Columba (Model)

The source code of Columba 2.0 was provided by my advisor of this thesis – Dr.-Ing. Tsun-Ming Tseng. Columba 2.0 was originally built for Linux system, and was planned for single execution. That is Columba stores its calculation results in a folder with the names from result_p1.txt to result_p3.txt. In other words, if someone tries to run Columba simultaneously, these temporary files will be overwritten, and the results will be incorrect.

To run multiple Columba programs on the server concurrently, we modify the original Columba and build it as a Windows application.

The first thing we have done was adding another parameter into the execution arguments of the "main" method. This parameter is used as a prefix which will be added to the front of the names of all saved files. In other words, result_p1.txt will be prefix_result_p1.txt whilst using this modified version. Now, to execute Columba, the arguments are:

```
Columba path-of-rule path-of-script p1-time p2-time p3-time p4-time prefix
```

Code 8: Execute Columba with these parameters

Secondly, we built the modified source code as a Windows application. To do that, we used Microsoft Visual Studio 2015 and imported all source code with needed dependencies, especially the Windows version of Gurobi Optimizer library, and by using the configuration given by Gurobi Optimizer's documentation (Gurobi Optimization, Inc., 2017), compiled and built a Windows executable. In the end, this executable can be used by our main controller.

5.3. Problem and Solution

One problem that occurred during testing was an issue of Gurobi Optimizer's license. As mentioned earlier, Columba invokes Gurobi Optimizer to solve the linear optimization problem. To execute Gurobi Optimizer, we used Gurobi Optimizer's academic license, which only allows one user per machine to run it (Gurobi Optimization, Inc., 2017). The user referred to here is the user account of a system, but not a real-world user. As a result, if our system's main user account is, for instance, "columbaServer", then only this user can run Gurobi Optimizer and thus Columba. Other users are not able to run them because of license restrictions, unless we have a commercial license, which allows multiple users to run the program.

Our main controller is running under Microsoft Internet Information Service (IIS). IIS has a built-in account, called IUSR, which executes the PHP script and, more importantly, Columba. That means, Columba can never be executed by our main controller with the normal user account, but will always be executed by the IIS built-in user account ISUR, unless we execute it manually. Thus, Columba cannot be invoked correctly by our main controller (a PHP script program) and will abort during execution, because of the license's user account is not IUSR.

The problem was that the built-in ISUR account is not a real user account, which means that we cannot directly login to that account and do a license registration for ISUR using the server. Thus, only scripts run on IIS uses this account. To solve that problem, we wrote a little PHP script, which registers an academic license and store it on the server. In that way, we can use the built-in user account IUSR to register it and have the license registered for IUSR.

```
system('echo d:|grbgetkey c1dxxxc-cdxx-de87-1c74-59xxxxxf9d36', $retval);
```

Code 9: Use PHP system function to register a license by using pipe

After that, we configured Gurobi Optimizer to use the new academic license. Finally, the Columba executed via PHP works without any issue, and designs are generated instantly.

6. Outlook and Summary

6.1. Outlook

The implementation of Cloud Columba – a web-based microfluidic automated design generation platform was successful. In the future, this platform can be extended into a "real" service platform. In other words, it should have a database to store information, especially to be able to have user accounts with password protected. Hence users will have their own spaces, such as history designs. These will be stored on the database. Furthermore, a real dashboard including other information, for instance some notifications, can be extended in the future. A possible looking could be like this:

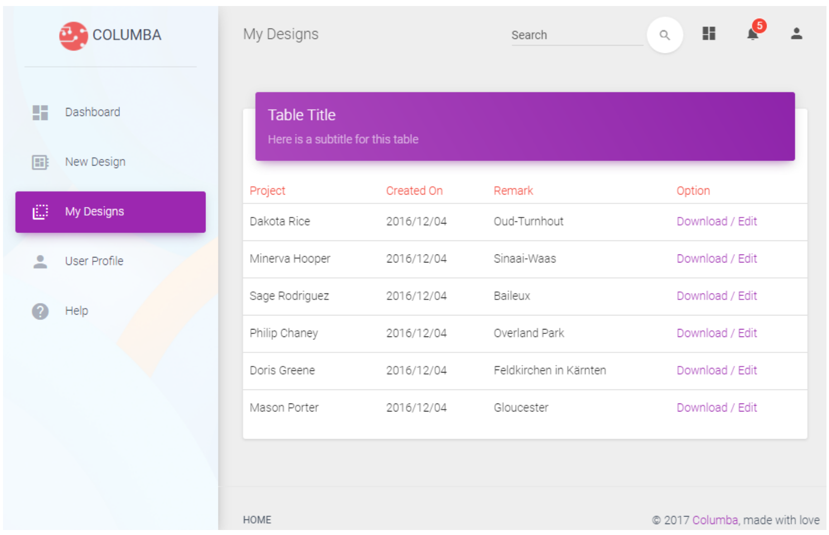


Table Title

Here is a subtitle for this table

Project	Created On	Remark	Option
Dakota Rice	2016/12/04	Oud-Turnhout	Download / Edit
Minerva Hooper	2016/12/04	Sinaai-Waas	Download / Edit
Sage Rodriguez	2016/12/04	Baileux	Download / Edit
Philip Chaney	2016/12/04	Overland Park	Download / Edit
Doris Greene	2016/12/04	Feldkirchen in Kärnten	Download / Edit
Mason Porter	2016/12/04	Gloucester	Download / Edit

HOME

© 2017 Columba, made with love

Figure 13: Proposal of possible extension

It can even be extended to a full online design tool for microfluidics. Specifically, users can drag and drop components such as mixers or chambers and define their sizes and locations. With such a visual interactive tool, scientists would not need any understandings of how to write a design script for Columba. With just simple clicks, they can design a microfluidic biochip.

6.2. Summary

In conclusion, this work went through the entire process of the implementation of Cloud Columba – a web-based microfluidic automated design generation platform, from system design up to final implementation.

The entire system consists of two parts: the back-end and the front-end.

The back-end has two parts as well. The first part is written in PHP and executes the second part written in C++, namely a modified Columba, after receiving data from the front-end, and then convert the generated design into SVG drawing for preview purposes. The front-end is written in HTML5, CSS3 and JavaScript. It uses a Responsive Web Design templet Metal Dashboard and Bootstrap framework. The front-end programs are written in JavaScript using JQuery library. The main function of the front-end is to show a user interface and transmit the input to the server for generating designs. After that, the design and its preview will be shown.

A. Code Examples

```
if ($_REQUEST["name"] || $_REQUEST["p1Time"] || $_REQUEST["p2Time"] || $_REQUEST["p3Time"] ||
$_REQUEST["rules"] || $_REQUEST["script"] || $script = $_REQUEST["p4"]) {
    $name = $_REQUEST["name"];
    $p1Time = $_REQUEST["p1Time"];
    $p2Time = $_REQUEST["p2Time"];
    $p3Time = $_REQUEST["p3Time"];
    $rules = $_REQUEST["rules"];
    $script = $_REQUEST["script"];
    $p4 = $_REQUEST["p4"];

    echo runColumba($name, $p1Time, $p2Time, $p3Time, $p4, $rules, $script);
}

function runColumba($name, $p1Time, $p2Time, $p3Time, $p4, $rules, $script) {
    global $COLUMBA_PATH;
    global $PATH;
    global $COLUMBA_EXE;
    global $COLUMBA_TRANSFORM_EXE;
    global $COLUMBA_RESULT_PATH;
    global $INPUT_PATH;
    global $SCR_OUPUT_PATH;

    $scriptFile = fopen($INPUT_PATH . $name . "_script.txt", "w") or die("Unable to open file!");
    $ret = fwrite($scriptFile, $script);
    fclose($scriptFile);

    $ruleFile = fopen($INPUT_PATH . $name . "_rule.txt", "w") or die("Unable to open file!");
    fwrite($ruleFile, $rules);
    fclose($ruleFile);

    $exec_cmd = $COLUMBA_EXE . " " . escapeshellarg($INPUT_PATH . $name . "_rule.txt") . " " .
escapeshellarg($INPUT_PATH . $name . "_script.txt") . " " . $p1Time . " " . $p2Time . " " . $p3Time . " " . $p4 . "
" . escapeshellarg($name);
    $transf_cmd = $COLUMBA_TRANSFORM_EXE . " " . $COLUMBA_RESULT_PATH . $name . "_result_p3.txt
" . $SCR_OUPUT_PATH . $name . ".scr";
    $exRt = exec($exec_cmd, $o, $ec); //$ec is exitcode
    if (substr($exRt, 0, 5) == "ERROR" || $ec != 0) {
        return "ERROR";
    }
    exec($transf_cmd);
    return $SCR_OUPUT_PATH . $name . ".scr";
}
```

Back-end Code: Receive data from front-end and execute Columba

```
function getSVGArcRadius($xa, $ya, $xb, $yb, $xc, $yc) {
    if ($xb == $xa) {
        if ($yc == $ya) {
            return abs($yb - $ya) / 2;
        } else {
            $yo = (1 / 2) * ($ya + $yb);
            $xo = $xa - (($yc - $ya) * ($yb - $ya)) / (2 * ($xc - $xa));
            return sqrt(($xo - $xa) * ($xo - $xa) + ($yo - $ya) * ($yo - $ya));
        }
    } elseif ($ya == $yb) {
        if ($xc == $xa) {
            return abs($xb - $xa) / 2;
        } else {
            $xo = (1 / 2) * ($xa + $xb);
            $yo = $ya - (($xc - $xa) * ($xb - $xa)) / (2 * ($yc - $ya));
            return sqrt(($xo - $xa) * ($xo - $xa) + ($yo - $ya) * ($yo - $ya));
        }
    } elseif ($yc == $ya && $xa != $xb) {
        $yd = 0.5 * ($ya + $yb);
        $xd = 0.5 * ($xa + $xb);
        $yo = $yd - (($yb - $ya) / ($xb - $xa)) * ($xa - $xd);
        return abs($yo - $ya);
    } elseif ($xa == $xc && $ya != $yb) {
        $yd = 0.5 * ($ya + $yb);
        $xd = 0.5 * ($xa + $xb);
        $xo = $xd - (($yb - $ya) / ($xb - $xa)) * ($ya - $yd);
        return abs($xo - $xa);
    } elseif ($xa != $xc && $xa != $xb && $ya != $yc && $ya != $yb && (($yc - $ya) / ($xc - $xa)) * (($yb -
$ya) / ($xb - $xa)) == -1) {
        return sqrt(($xb - $xa) * ($xb - $xa) + ($yb - $ya) * ($yb - $ya)) / 2;
    } else {
        $yd = ($ya + $yb) / 2;
        //o is the center of this curve
        $xo = ($yd - $ya + (($xb - $xa) / ($yb - $ya) - ($xc - $xa) / ($yc - $ya)) * $xa) /
                (($xb - $xa) / ($yb - $ya) -
                ($xc - $xa) / ($yc - $ya));
        $yo = $ya - (($xc - $xa) / ($yc - $ya)) * ($xo - $xa);
        $radius = sqrt(($xo - $xa) * ($xo - $xa) + ($yo - $ya) * ($yo - $ya));
        return $radius;
    }
}
```

Back-end Code: Calculate radius for SVG arc description

```
function parseSCRArray($arr) {
        $current_layer = ""; //Current Layer
        $current_color = []; //Current Layer ACI in RGB
        $layer_colors = []; //Maping of Color definition for all layers [LayerName]=>[ACI]
        $svg_head  =  '<?xml  version="1.0"  encoding="UTF-8"?><svg  xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" height="';
        $svg_tail = '</g></svg>';
        $svg_body = '<g id="group1">';
        $maxXY = "0,0"; //Max X and Y point, to define pic size
        $path = ""; //Temp String for SVG <path >
        $arr_len = count($arr); //get Array Length
        $last_point = "0,0"; //Last point of PLINE part
        $stroke_width = "10";
        $fill = "none";
        for ($i = 0; $i < $arr_len; $i++) {
                if (substr($arr[$i], 0, 5) == "COLOR") {
                        //splitt the cmd into ACI and Layer
                        $color = substr($arr[$i], 6);
                        $clrNlayr = explode(" ", $color);

                        $layer_colors[trim($clrNlayr[1])] = $clrNlayr[0];
                } elseif (substr($arr[$i], 0, 6) == "LAYER ") {
                        //set the current layer and color
                        $layer = substr($arr[$i], 6);
                        $layr = explode(" ", $layer);
                        if ($layr[0] === "S") { //only reacts on LAYER S ***
                                $current_layer = trim($layr[1]);
                                //echo print_r($layer_colors);
                                $current_color = ACItoRGB($layer_colors[$current_layer]);
                                $stroke = "rgb(" . $current_color[0] . "," . $current_color[1] . "," . $current_color[2] . ")";
                                $svg_body = $svg_body . '</g> <g id="' . $current_layer . '" stroke = "' . $stroke . '" stroke-
width="' . $stroke_width . '" fill ="' . $fill . '">';
                        }
                } elseif (substr($arr[$i], 0, 5) == "PLINE") {
                        $path = '<path d="';
                        $i++;
                        $path = $path . "M" . $arr[$i] . " ";
                        $maxXY = testGreaterXY($maxXY, $arr[$i]);
                        $last_point = $arr[$i];
                        while (is_numeric(substr($arr[$i + 1], 0, 1))) {
                                $i++;
                                $path = $path . "L" . $arr[$i] . " ";
                                $last_point = $arr[$i];
                        }
                } elseif (substr($arr[$i], 0, 5) == "Arc D") {
                        $i++;
                        $arc_c = $arr[$i]; //Arc Point C (Start Point Vector)
                        $i++;
                        $arc_b = $arr[$i]; //Arc Point B (End point)
                        $xyRadius = getSVGArcRadius(getX($last_point), getY($last_point), getX($arc_b), getY($arc_b),
getX($arc_c), getY($arc_c));
                        $sweepFlag    =    getSVGArcSweep(getX($last_point),    getY($last_point),    getX($arc_b),
getY($arc_b), getX($arc_c), getY($arc_c));
                        $path = $path . "A" . $xyRadius . "," . $xyRadius . "," . "0,0," . $sweepFlag . "," . $arc_b . " ";
                        $last_point = $arr[$i];
                } elseif (substr($arr[$i], 0, 4) == "Line") {
                        $i++;
                        $path = $path . "L" . $arr[$i] . " ";
                        $last_point = $arr[$i];
                        while (is_numeric(substr($arr[$i + 1], 0, 1))) {
                                $i++;
                                $path = $path . "L" . $arr[$i] . " ";
                                $last_point = $arr[$i];
                        }
                } elseif (substr($arr[$i], 0, 5) == "Close") {
                        $path = $path . 'Z"    />';
                        $svg_body = $svg_body . $path;
                } else {
                        $svg_body = $svg_body . "<!-- " . $arr[$i] . " -->"; //Other Command insert as Comment
                }
        }
        // unset($cmd);

        $svg_head = $svg_head . (intval(getY($maxXY)) + 100) . '" width="' . (intval(getX($maxXY)) + 100) . '">';
//update svg size with maxXY
        $svg_flip_head = '<g transform="translate(0,' . (intval(getY($maxXY)) + 100) . ') scale(1, -1)">';
        $svg_flip_tail = "</g>";
        return $svg_head . $svg_flip_head . $svg_body . $svg_flip_tail . $svg_tail;
}
```

Back-end Code: Translate AutoCAD script into SVG

```
//=========================================================================
// Click Open File Dialog Functions
//=========================================================================
function openRule()
{
	document.getElementById("openRule").click();
}

function readRuleFile(e) {
	var file = e.target.files[0];
	if (!file) {
		return;
	}
	var reader = new FileReader();
	reader.onload = function (e) {
		var contents = e.target.result;
		displayRules(contents);
	};
	reader.readAsText(file);
}

function displayRules(contents) {
	var element = document.getElementById('rules');
	element.value = contents;
}

function openScript()
{
	document.getElementById("openScript").click();
}

function readScriptFile(e) {
	var file = e.target.files[0];
	if (!file) {
		return;
	}
	var reader = new FileReader();
	reader.onload = function (e) {
		var contents = e.target.result;
		displayScript(contents);
	};
	reader.readAsText(file);
}
```

```
function displayScript(contents) {
	var element = document.getElementById('script');
	element.value = contents;
}

//=========================================================================
// Save File Function
//=========================================================================
function saveText(sourceID, ext) {
	var script = $("#" + sourceID).val();
	var blob = new Blob([script], {type: "text/plain;charset=utf-8"});
	saveAs(blob, sourceID + "." + ext);
}
function saveSVG(data, ext) {

	var blob = new Blob([data], {type: "image/svg+xml;charset=utf-8"});
	saveAs(blob, "design ." + ext);
}

//=========================================================================
// Nav Bar Select Content Functions
//=========================================================================
var jumpContents = "";
var jumpID = "";
var noWarning = false;
function selectContent(contents, id) {
	var x = location.hash;
	if ("#" + id === x) {
		return;
	}
	if (siteHasForm() && !noWarning) {
		jumpContents = contents;
		jumpID = id;
		openLeaveWarnModal();
		return;
	}
	unactiveNav();
	$("#" + id).attr("class", "active");
	$("#content").load(contents);

	var title;
	location.hash = "#" + id;
	switch (id) {
		case "dashboard":
```

```
                title = "Dashboard";
                break;
            case "new":
                noWaring = false;
                title = "New Design";
                break;
            case "user":
                noWaring = false;
                title = "User Profile";
                break;
            case "help":
                title = "Help";
                break;
            case "list":
                title = "My Designs";
                break;
            case "result":
                title = "Result";
                $("#new").attr("class", "active");
                break;
        }
        $("#contentTitle").text(title);
}

function openLeaveWarnModal() {
        $('#warnModal').modal('toggle');
}

function siteHasForm() {
        var x = location.hash;
        return (x === "#new" || x === "#user");
}
function unactiveNav() {
        $("#dashboard").removeAttr("class");
        $("#user").removeAttr("class");
        $("#new").removeAttr("class");
        $("#help").removeAttr("class");
        $("#list").removeAttr("class");
}
function leavePage() {
        location.hash = "#";
        selectContent(jumpContents, jumpID);
}
```

```
//=======================================================================
// Script Data Processing and Posting
//=======================================================================
function generateDesign() {
    noWarning = true;
    if (newValidate) {
        postData();
    }
}
function postData() {
    var scr_File = "";
    var svg_Element = "";
    $('#generateBtn').prop('disabled', true);
    $('#generateBtn').text("Generating...")
    $('#script').prop('disabled', true);
    $('#projectName').prop('disabled', true);
    $('#p1Time').prop('disabled', true);
    $('#p2Time').prop('disabled', true);
    $('#p3Time').prop('disabled', true);
    $('#p4').prop('disabled', true);
    $('#rules').prop('disabled', true);
    $('#pleaseWait').prepend('<div class="alert alert-success" role="alert"><h4 class="alert-heading">Your Design Is
Generatng...</h4><p>You will be redirected shortly, please wait a moment.</p></div>');
    $("html, body").animate({scrollTop: 0}, "slow");

    $.post("newGenerate.php",
            //Data
                {
                    name: $("#projectName").val(),
                    p1Time: $("#p1Time").val(),
                    p2Time: $("#p2Time").val(),
                    p3Time: $("#p3Time").val(),
                    p4: $("#p4").val(),
                    rules: $("#rules").val(),
                    script: $("#script").val()
                },
                //Server Response function
                    function (data, status) {
                        scr_File = data;
                        //alert("Data: " + data + "\nStatus: " + status);
                        if (data === "ERROR") {
                            $('#pleaseWait').html('<div class="alert alert-danger" role="alert"><h4
class="alert-heading">Error!</h4><p>An error was occured. Please check your inputs.</p></div>');
                            $('#generateBtn').prop('disabled', false);
                            $('#generateBtn').text("GENERATE DESIGN")
                            $('#script').prop('disabled', false);
                            $('#projectName').prop('disabled', false);
                            $('#p1Time').prop('disabled', false);
                            $('#p2Time').prop('disabled', false);
                            $('#p3Time').prop('disabled', false);
                            $('#p4').prop('disabled', false);
                            $('#rules').prop('disabled', false);
                            return;
                        }
                        selectContent("/contents/result.php?FileName=" + data, "result");
            });


                //svgCanvas.setSvgString(svg_Element);
            }
```

Front-end Code: JavaScript utility functions

```
            <div class="col-md-8">
                <div class="card card-nav-tabs">
                    <div class="card-header" data-background-color="orange">
                        <h4 class="title">Script</h4>
                        <p class="category">Edit your design script here</p>
                    </div>
                    <div class="card-content">

                        <!-- with icons and horizontal -->
                        <ul class="nav nav-pills nav-pills-icons nav-pills-primary">
                            <li>
                                <a    onclick="openScript(); return;">
                                    <i class="material-icons">folder_open</i>
                                    Open<input type="file" id="openScript" style="display:none"
name="script" accept=".clbs,.txt,text/txt,txt/clbs">
                                </a>
                            </li>
                            <li>
                                <a    onclick="saveText('script', 'clbs')">
                                    <i class="material-icons">save</i>
                                    Save
                                </a>
                            </li>
                            <li>
                                <a href="#tasks">
                                    <i class="material-icons">list</i>
                                    Syntax
                                </a>
                            </li>
                        </ul>
                        <div class="row">
                            <div class="col-md-12">
                                <div class="form-group">
                                    <label>Code Editor</label>
                                    <div class="form-group label-floating">
                                        <label class="control-label">&nbsp;      &nbsp;Script goes
here </label>
                                        <pre class="prettyprint prettyprinted"><textarea
class="form-control" rows="34" id="script" required></textarea></pre>
                                                        </div>
                                                    </div>
                                                </div>
                                            </div>
```

```
                                                        <div class="clearfix"></div>

                                                </div>
                                        </div>
                        <button type="submit" class="btn btn btn-success pull-right"
id="generateBtn">Generate Design</button>
                        <!--<button type="submit" class="btn btn btn-success pull-right"
onclick="generateDesign();">Generate Design</button>-->
                                </div>
```

Front-end Code: HTML Layout

Bibliography

Bootstrap, 2017. *Bootstrap · The most popular HTML, CSS, and JS library in the world.* [Online]
Available at: http://getbootstrap.com
[Accessed 26 08 2017].

Gurobi Optimization, Inc., 2017. *GUROBI OPTIMIZER.* [Online]
Available at: http://www.gurobi.com/documentation/current/quickstart_linux.pdf
[Accessed 24 08 2017].

Hu, K., Chakrabarty, K. & Ho, T.-Y., 2017. Design Automation. In: *Computer-Aided Design of Microfluidic Very Large Scake Integration Biochips.* s.l.:Springer International Publishing AG, pp. 16-17.

Hu, K., Dinh, T., Ho, T.-Y. & Chakrabarty, K., 2017. Control-layer routing and control-pin minimization. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* Issue 36, pp. 55-68.

JQuery, 2017. *jQuery: The Write Less, Do More, JavaScript Library.* [Online]
Available at: https://jquery.com/
[Accessed 15 06 2017].

Minhass, W. & P. Pop, J. M., 2011. System-level modeling and synthesis of flow-based microfluidic. *Proceedings International Conference on Compilers, Architectures and Synthesis*, pp. 225-233.

Moses, 2017. *AutoCAD Color Index (ACI) RGB equivalents.* [Online]
Available at: http://sub-atomic.com/~moses/acadcolors.html
[Accessed 15 06 2017].

Potel, M., 1996. *MVP: Model-View-Presenter.* [Online]
Available at: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf
[Accessed 27 08 2017].

Schade, A., 2014. *Nielsen Norman Group.* [Online]
Available at: https://www.nngroup.com/articles/responsive-web-design-definition/
[Accessed 29 08 2017].

Stanford Microfluidic Foundry, 2017. *Basic Design Rules.* [Online]
Available at: http://www.stanford.edu/group/foundry/Basic%20Design%20Rules.html
[Accessed 27 08 2017].

Syazwan, A. et al., 2017. *SVG-edit is a fast, web-based, JavaScript-driven SVG drawing editor.* [Online]
Available at: https://github.com/SVG-Edit
[Accessed 15 06 2017].

Tim, C., 2017. *Material Dashboard.* [Online]
Available at: https://www.creative-tim.com/product/material-dashboard

[Accessed 15 06 2017].

Tseng, T.-M. et al., 2017. [Under Review] Columba 2.0: A Co-Layout Synthesis Tool for Continuous-Flow Microfluidic Biochips. *IEEE Trans. on Comput.-Aided Des. of Integr. Circuits and Syst.*

W3Techs, 2017. *Usage Statistics and Market Share of Web Servers for Websites, August 2017.* [Online]
Available at: https://w3techs.com/technologies/overview/web_server/all
[Accessed 27 08 2017].

Wikipedia, 2017. *Wikipedia - Responsive web design.* [Online]
Available at:
https://upload.wikimedia.org/wikipedia/commons/thumb/0/02/Complete.png/285px-Complete.png
[Accessed 29 08 2017].