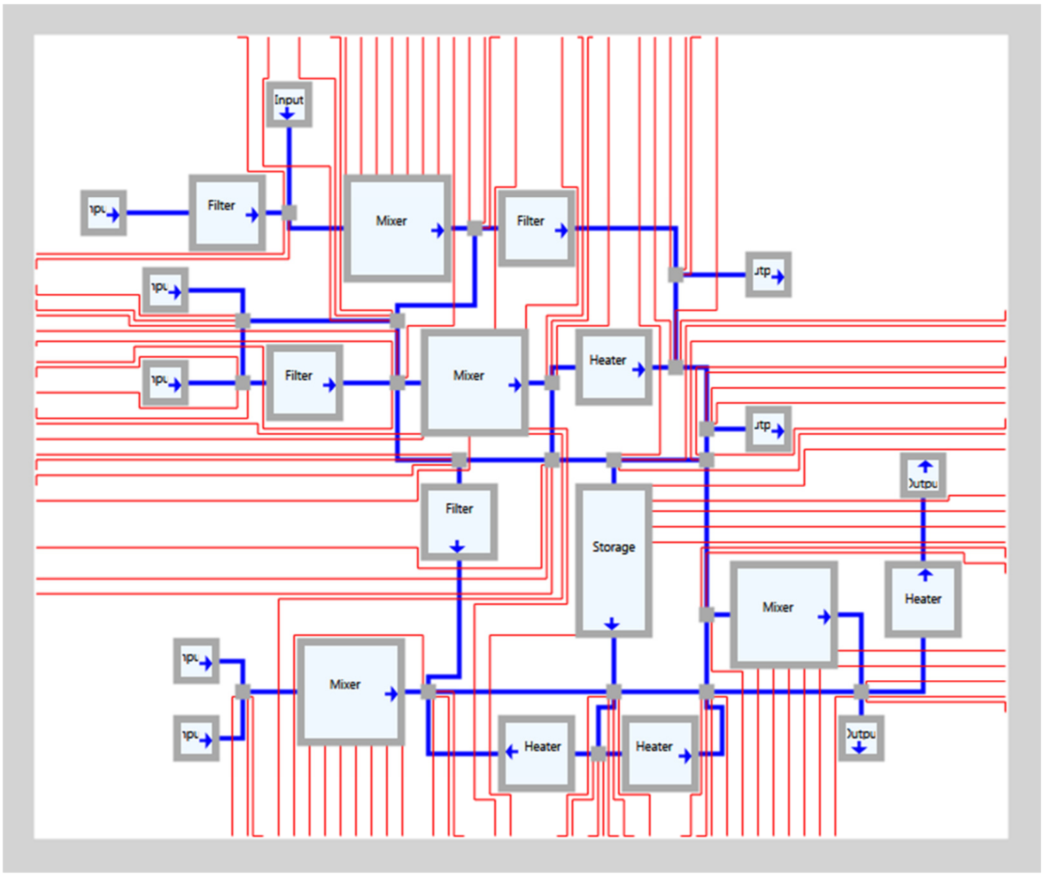
Routing Algorithms for Flow-based Microfluidic Very Large Scale Integration Biochips



28. June 2013

By:

Martin Simonsen Hørslev-Petersen, S103054

Thomas Onstrup Risager, S103040

Supervisor:

Paul Pop, DTU Compute

Abstract

Microfluidic Very Large Scale Integration (mVLSI) is the technology behind the creation of continuous flow-based biochips. By combining miniaturized valves and channels it is possible to create various components that can be used to manipulate fluids, thus creating a microfluidic laboratory on a chip.

Designers are currently using drawing tools such as AutoCAD to manually place components and route channels. With the increase in the complexity of the biochips, such manual approaches become infeasible.

The objective of this thesis is to develop automatic routing tools for the routing of the flow and control channels of mVLSI biochips. First, this bachelor thesis investigates different grid based routing algorithms previously suggested for the Very Large Scale Integration (VLSI) of microelectronic chips. The idea is to identify and implement candidates that might solve the related problems of microfluidic Very Large Scale Integration.

For the grid based routing problem we studied Lee's algorithm and Hadlock's algorithm for routing. These two algorithms had to be extended to accommodate the design requirements of flow-based microchips.

The analysis and implementation of the algorithms revealed some weaknesses with regards to Hadlock's algorithm. Firstly, the Hadlock algorithm requires a single specific target in order to calculate its detour values correctly. This means that routing to the closest of a collection of several different points is simply not possible. Secondly, the Hadlock algorithm does not guarantee to find the shortest path between a terminal and several starting points. Therefore, in our opinion, these weaknesses made Hadlock's algorithm inferior to Lee's algorithm.

As a result of the above mentioned issues, we have integrated and extended the Lee algorithm, as part of our custom Computer Aided Design (CAD) tool that helps and guides the user through the process of designing a continuous flow-based biochip. Furthermore, we have implemented a simulated annealing placement algorithm, which is also integrated into the tool. We have evaluated the CAD tool on several case studies.

1 Introduction

Different biochips, which are alternatives to the conventional biochemical laboratories, have emerged over the past few years; some of them may have the potential to revolutionize life sciences. Microfluidic biochips decreases the reagent consumption and increases the throughput compared with traditional laboratory work [1], and biochips have been applied in many areas [2]. Two of the main alternatives are continuous flow-based microfluidic biochips and digital microfluidic biochips [3].

Digital microfluidic biochips, or droplet-based biochips, use a two dimensional array of electrodes to manipulate droplets of different fluids. The droplets are moved, mixed and split using electric currents. Afterwards, various properties can be detected in the manipulated droplets [2].

Continuous flow biochips are custom biochips designed for a given task. This is done by manipulating liquid, not as droplets, but as a continuous flow. An example of a continuous flow biochip is presented in Figure 1. These kinds of biochips consist of different layers, which generally can be divided into two categories – A flow layer and a control layer. As one would imagine the liquid flows through the flow layer, within small closed channels, and are controlled by the control layer using air pressure. The biochips are mostly fabricated using soft lithography, which is a cheap rubber-like transparent material.

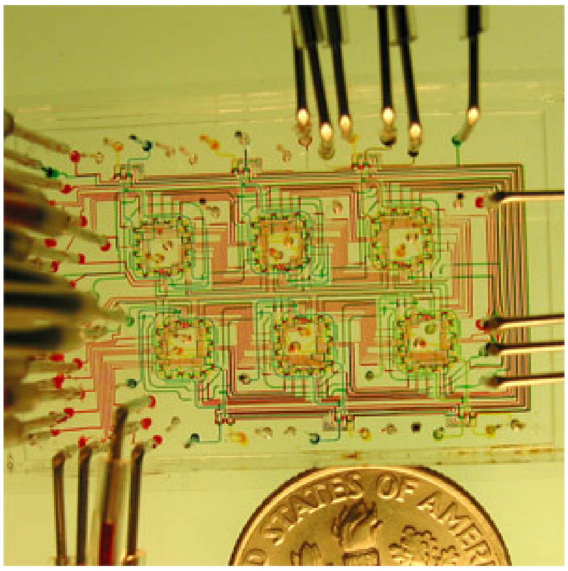


Figure 1: Flow-based microfluidic biochip
Borrowed from [4]

At present, the main obstacle that is holding back advancement in the field of microfluidic Very Large Scale Integration of continuous flow-based biochips is the labor intensive and often error prone manual design process. Although some Computer Aided Design (CAD) tools are emerging [2], the design and fabrication process is still mostly manual. The design process requires extensive knowledge of the application of the chip being designed and the tools being used.

mVLSI somewhat resembles the related problem of Very Large Scale Integration (VLSI), which is the process of designing large integrated electronic circuits. In this paper we will analyze different algorithms, which may support the specialized labor-intensive continuous flow-based biochip design process and we will look into different grid-based algorithms, which previously have been suggested for solving VLSI and other similar problems.

Furthermore, the aim is to implement a complete custom CAD tool that may aid the designer with the entire process from designing to placement and routing.

2 Motivation

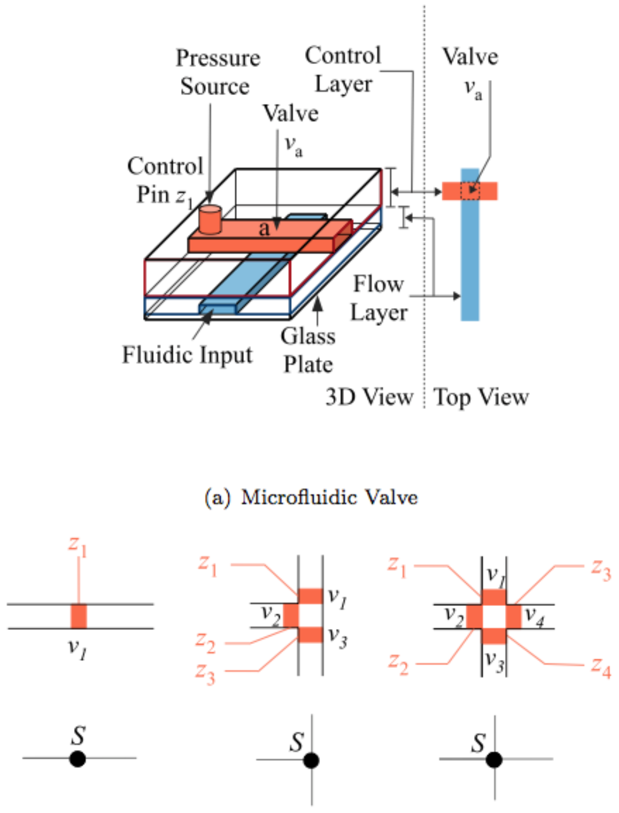
As already mentioned the process of designing microfluidic flow-based biochips is highly labor intensive. It requires extensive knowledge of the application and the currently available design tools. The designer currently relies on various component templates and a set of design rules. This is currently done using drawing tools such as AutoCAD, by manually placing the component templates and connecting them by drawing lines representing the liquid flow channels. The designer needs to constantly ensure that the different design rules are being met. Furthermore, as the size and complexity of the biochips increases then so does the difficulty and complexity of the design process. This means that it is both expensive and time consuming to create new designs and therefore it is desirable to try to automate some if not all of the design process.

3 Continuous Flow-based Biochip Architecture

Flow-based biochips are constructed from a single flow layer and one or two control layers.

The flow layer consists of the components, which are used to manipulate fluids, and flow channels through which liquids are transported to and from the components. The liquids are introduced into the biochip via tubes connected to inlet ports that are placed at various points on the biochip. Since the flow channels are allowed to intersect each other, the need to control the flow of the liquids through the flow channels on the biochip is introduced.

When a liquid is moving from one component to another it will have to be guided through the various intersections, to end up at the right place. This is achieved by the use of either one or two control layers that may be configured as either a single control layer, as illustrated below on Figure 2a, or as two separate control layers, one above and one below the flow layer. These control layers consists of either several push-down or push-up valves which are connected to high pressure air control channels. The valves are usually connected to external high pressure air sources, which may be connected to entry points on the edge of the chip.



(a) Microfluidic Valve

(b) Switch Configurations

Figure 2: Microfluidic Flow-based valve and switch.
Borrowed from [2].

The valves controls the liquids flowing through the flow layer by applying pressure to a membrane between the control layer and the flow layer, thus creating a tight seal that stops the liquid from flowing. For example, routing a liquid from south to north can be done by applying air pressure to valves 2 and 4 in the last (right) switch in Figure 2b, thus closing the valves of the east and west flow channels. In addition, this 4-direction switch requires up to 4 independent control lines, whereas the middle T-cross switch requires a maximum of 3.

This way of controlling the liquids, by the use of air pressure, creates different constraints for the routing of the control channels. Since the high pressure air in a particular control channel is distributed uniformly, only valves that are open and closed at the same time can share a control channel.

Furthermore, valves are used inside various components for the operation of these components. Two examples that we have included in this paper are the Mixer component and the Storage component.

For the Mixer, the mixing of fluids is performed using valves placed inside the component, by activating and de-activating the valves in a certain order, to move the fluids within the component around. For the storage unit, liquids are directed into specific channels, and stored there until they are needed. The liquids are kept in place by activating valves with the use of air pressure.

The relative sizes of the valves needed to control the components can be seen below on Figure 3.

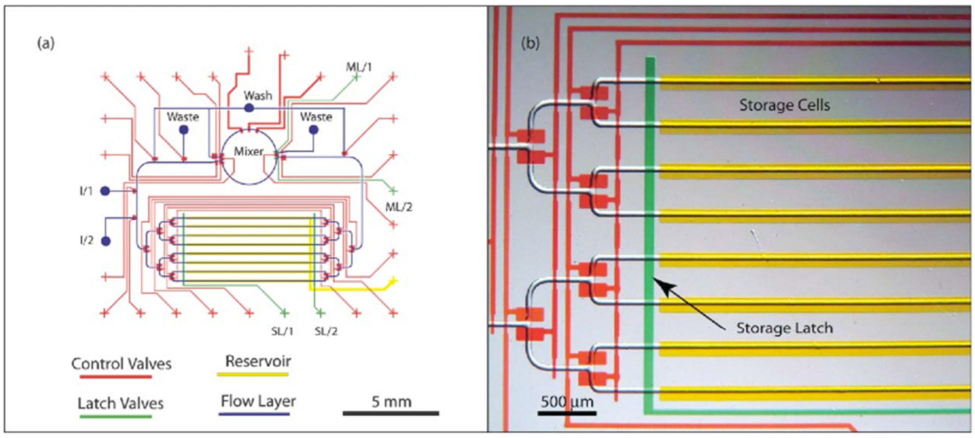


Figure 3: Actual sizes of the components on a biochip.
Borrowed from [1].

4 Problem description

The process of designing a biochip consists of a number of different steps. These steps are:

- System specification
- Schematic design
- Placement
- Flow channel routing
- Application mapping
- Control synthesis
- Control channel routing
- Fabrication

During our work we have concentrated on the steps: Placement, Flow channel routing, and Control channel routing. For a detailed description of these steps, please see [2].

4.1 System specification and Schematic design

The design process of a flow-based biochip starts with a given system specification, upon which an initial schematic design can be based.

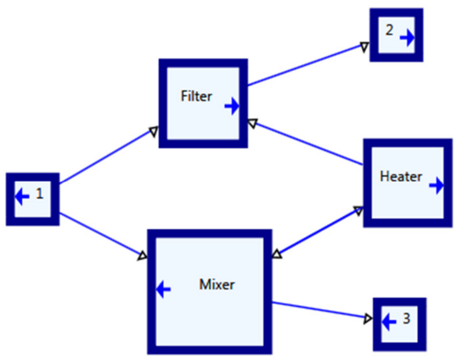


Figure 4: Simple schematic design.

Figure 4 above shows an example of some individual components and their connections to other components. This schematic design could be described as shown in Table 1 and Table 2 in tabular form.

Components				
ID	Name	Type	Height	Width
1	1	Input	3	3
2	2	Output	3	3
3	3	Output	3	3
4	Filter	Filter	5	5
5	Heater	Heater	5	5
6	Mixer	Mixer	7	7

Table 1: List of components, as shown on Figure 4.

Connections				
ID	From	From ID	To	To ID
1	1	1	Mixer	6
2	1	1	Filter	4
3	Filter	4	2	2
4	Heater	5	Mixer	6
5	Heater	5	Filter	4
6	Mixer	6	Heater	5
7	Mixer	6	3	3

Table 2: List of connections, as shown on Figure 4.

4.2 Placement

During the placement part of the design process it is decided where the components of the chip are to be placed. The placement has not been the focus of our thesis, which is about routing, but we need to know the placement before the routing can be done. There are several key elements that should be taken into account during the placement process.

The placement of the components should be done so as to minimize the chip size, and the length of the flow and control channels respectively. The placement phase should furthermore ensure that the routing of flow and control channels is actually possible.

The optimal approach would be to solve the placement and the routing problems simultaneously. However, this approach is not feasible due to the amount of required computations. It is therefore necessary to split these two processes.

The splitting of the two highly dependent processes now introduces a new problem, which is that a way to evaluate a given placement is needed. For example, summing the Euclidian distances between the connected objects could be used as a relative measure when comparing different placements. Then, the placement with the smallest summarized Euclidian distance would be the better solution.

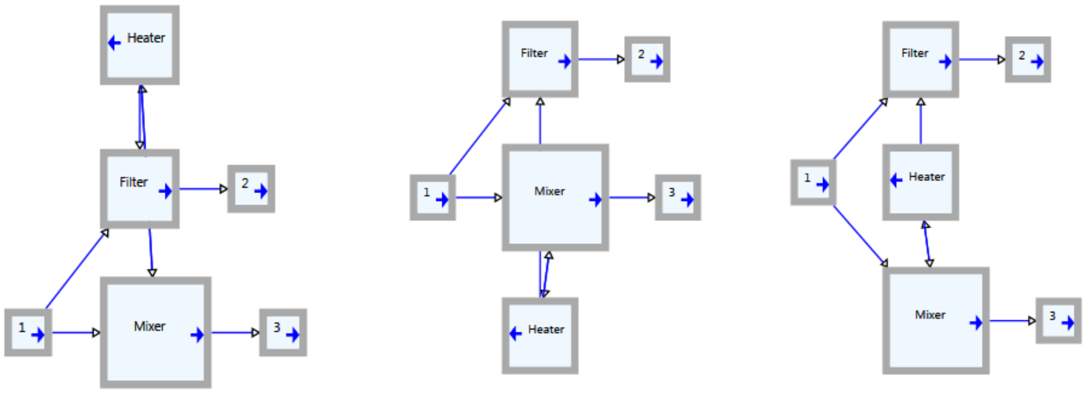


Figure 5: Manually done placement solutions of the design shown on Figure 4.

The three different placement solutions presented above in Figure 5 are all good placement solutions done manually. They are arranged from left to right based on each of their summarized Euclidian distances with the left most being the worst. We will later return to the solutions above and create different flow and control routings based upon them.

There are many VLSI placement algorithms proposed in the literature. We have selected a Simulated Annealing placement algorithm [5] to produce simple placement solutions, needed as an input to our routing problem. The algorithm uses the Euclidian distances as a cost function.

The schematic design shown below on Figure 6 is again based on the schematic design shown in Figure 4, but this one is created solely using our simulated annealing algorithm in our custom CAD tool.

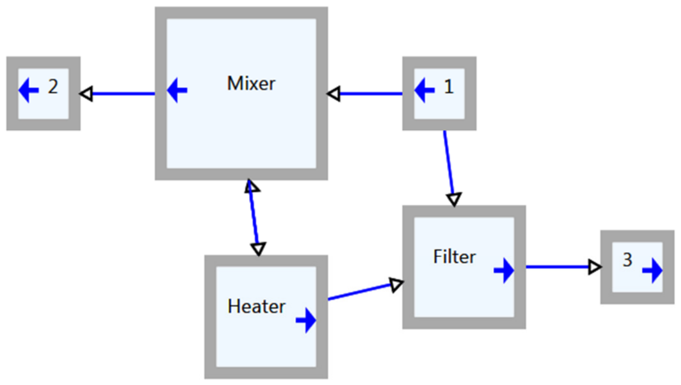


Figure 6: Automated CAD placement of the design shown on Figure 4.

4.3 Flow Routing

The flow routing part of the design process is where the flow channels connecting the components are drawn. These channels may be allowed to cross each other by introducing non-mandatory intersections, if this is considered most optimal. As previously mentioned, the aim is to make the flow channels that are connecting the components as short as possible while at the same time minimizing the number of non-mandatory intersections. Because these two parameters affect each other adversely, the goal will be to find a certain balance between them for meeting the requirements of the design.

The components on a biochip all have a single input port and/or a single output port. Therefore the first problem that must be solved is to figure out how to connect multiple flow lines to single ports. To visualize this, let us consider the simple design illustrated in the left part of Figure 7.

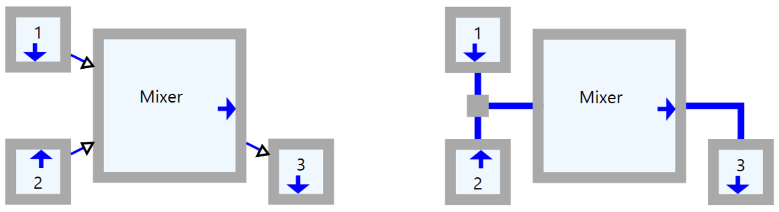


Figure 7: Simple placement containing mixer with multiple inputs (left), and the same design after routing (right).

As can be seen on the placement, the Mixer is fed from the two Inlet Terminals 1 and 2. But because the Mixer only has a single input port, the flow lines coming from the output ports of each of the two Inlet Terminals must be joined together and then connected to the input port of the Mixer. To summarize, what needs to be joined together are:

- Output port of Inlet Terminal 1
- Output port of Inlet Terminal 2
- Input port of Mixer

A dedicated algorithm for solving these kinds of problems will be required, so as to convert the defined point-to-point flow channel connections into groups of points that must be connected.

Another problem that must be addressed is what should be done if a required flow line is cut off by another flow line. As an example, let us again consider the first (left) simple placement illustrated in Figure 5.

This design can be routed in numerous ways. Two possible solutions based on the design can be seen in Figure 8. Here, for the first (left) routing, the flow channel going from the Mixer to the Heater is routed around the flow channel going from the Filter to Outlet Terminal 2, whereas for the other (right) routing, an intersection has been introduced. This intersection is considered non-mandatory because it is not defined in the schematic design. In other words, it is not meant to be there according to the schematic design.

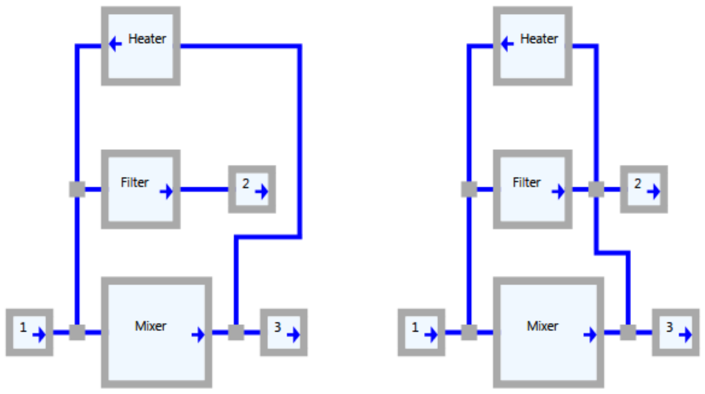


Figure 8: Routing with no non-mandatory intersections (left), and with a non-mandatory intersection (right).

The advantage of having the first solution is that the number of control lines is reduced. This is due to the fact that intersections require dedicated control lines for closing off the necessary valves inside the intersection, to prevent fluids from one channel to flow into the intersecting channel. The valves associated with an intersection or switch can be seen on Figure 2b. This means that the additional intersection, introduced in the second (right) flow channel routing on Figure 8, requires an additional 4 control channels. Also, because the channels do not intersect each other, parallel operation of the two flow lines is possible, which may also be an advantage. The drawback is that one of the flow lines is very long which will increase the time it takes for fluid to move between the components connected by this line. Furthermore in some cases the 'route-around' solution will introduce additional turns to the flow path, which is also not desirable.

In the other (right) design the advantage is that both flow lines are of optimal length, which will facilitate efficient transfer of fluid between components. The drawback is that fluid will have to travel through a non-mandatory intersection, which will inhibit parallel operation, and also additional control lines for this intersection must be included in the design, for the reasons mentioned above.

The logic that decides which solution should be picked must be incorporated into the flow routing algorithm.

4.4 Application Mapping

The next phase of the design process is to do the mapping of the application onto the chip, based on the attained flow routing design. As the lengths of the different flow channels between the components are now known, the flow latencies of the liquids flowing through these channels can be calculated. This means that the execution schedule for the given application can be created.

4.5 Control Synthesis

Knowing the execution schedule will allow us to identify the valves that must be activated at various execution times. This in turn will allow us to minimize the number of needed valves since certain valves might be able to share the same air pressure inlet port.

In this paper we will not go into the details regarding the application mapping and control synthesis. But the fact that single control channels may be connected to several control valves will be incorporated in the design of our control channel routing algorithm.

4.6 Control Routing

The control routing part of the design process is where the control channels, connecting the microfluidic valves with the high pressure inlets, are drawn. Control channels are not allowed to intersect. If they did intersect, then the high pressure air from one control channel would propagate into all the intersecting channels and then activate all the valves connected to these channels. In the example shown on the left side of Figure 9, we have connected four components to a single control channel and two of the components to another single control channel. Now, if the heater in the middle needs to be connected to its own high pressure air source, which would be a third control channel, then a way to determine which control line(s) to rip up is needed.

In our control channel routing algorithm, we have implemented a rip-up and re-route scheme [7]. The main problem with a rip-up and re-route scheme is deciding which nets to rip up, for efficiently solving conflicts between nets. For this reason, we have developed a control channel routing algorithm that is able to intelligently identify nets that must be ripped up. The algorithm is based on the same Lee and back tracking algorithms that we implemented for the flow channel routing algorithm, but with a few key changes. These changes will be described in the chapter about algorithmic design.

A possible solution of the problem shown on the left side of Figure 9 is show on the right side of the figure. Here, the routing allows for the heater to connect to an inlet port.

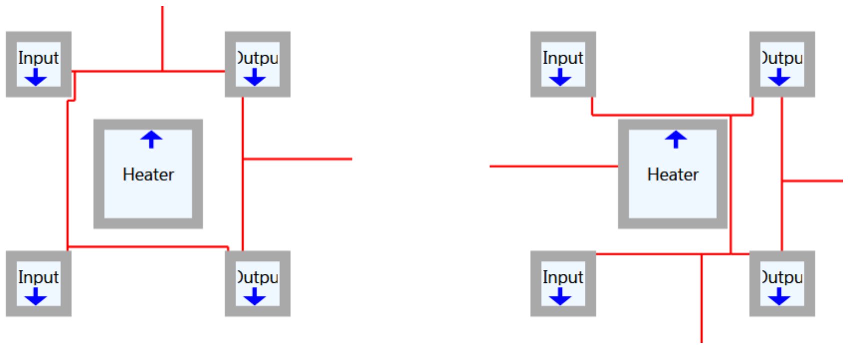


Figure 9: Heater boxed in by control channel routings (left). The same problem solved by rip-up and re-routing (right).

5 VLSI and possible algorithms

Several different algorithms have been suggested for routing VLSI chips [6]. In this section, we will look into some of the different possibilities that are based upon a grid structure.

Overall there are two different approaches to solving grid based routing problems, the sequential and the concurrent approach.

The sequential approach routes the paths of the various nets, as the name suggest, in a sequential way one net after the other. Because of this sequential approach previously routed nets might end up blocking the path of other nets. This means that the actual routing is highly dependent on the order of which the nets are routed. Again, different approaches have been suggested in order to solve the problem of nets blocking each other [6].

The first approach aims to assign criticality to the different nets that need to be routed. This means that the higher the criticality the more likely it is to be blocked by other nets. Because of this, nets with higher criticality should be routed first. In order to determine the criticality of a net one could look at the area on the grid that it covers or the number of terminals to be connected [6].

Another approach is based on a rip-up and re-route [7] idea. This means that if the routing of a net fails due to it being blocked, then some of the previously routed nets are ripped up. This should facilitate the routing of the blocked net and afterwards the nets that were ripped up are then re-routed. This rip-up and re-routing process has the disadvantage that it increases the running time of the routing algorithm, because some nets need to be routed more than once.

The last solution suggested is to incorporate a shove algorithm [7]. This algorithm tries to shove the paths of obstructing nets thus making room for other nets to be routed.

The concurrent approach avoids the problem of routed nets obstructing new nets by routing them concurrently. This means that all the nets are routed simultaneously thus making sure never to block the paths of other nets. The main problem with this approach is that it is computationally hard. In fact there exists no polynomial time grid-based algorithm for solving the problem concurrently [6]. In this paper we will not go into further detail with this concurrent approach. Instead we will look into the following three sequential grid based algorithms: Lee's algorithm, Hadlock's algorithm and Steiner Trees.

Before going into detail about the different algorithms we will introduce a simple graph structure equivalent to the grids used by these algorithms. The graph G=(V,E) could represent the grid used by the algorithms. The grid consists of square cells of a certain size arranged in an array of height h and length l. Each of the cells c_i represent a vertex v_i in graph G. Furthermore there exits an edge between two vertices v_i and v_j if and only if the two cells c_i and c_j are adjacent [6]. For our different examples, throughout this paper, we will be using the array-based visualization.

As a part of the description and the analysis of the different algorithms we will be using the example below, shown in Table 3. For the following tables we have color coded the cells with regards to what they contain to make it easy to tell them apart. Obstacles have been colored blue, previously routed control channels grey, and the newly routed control channel has been colored light red.

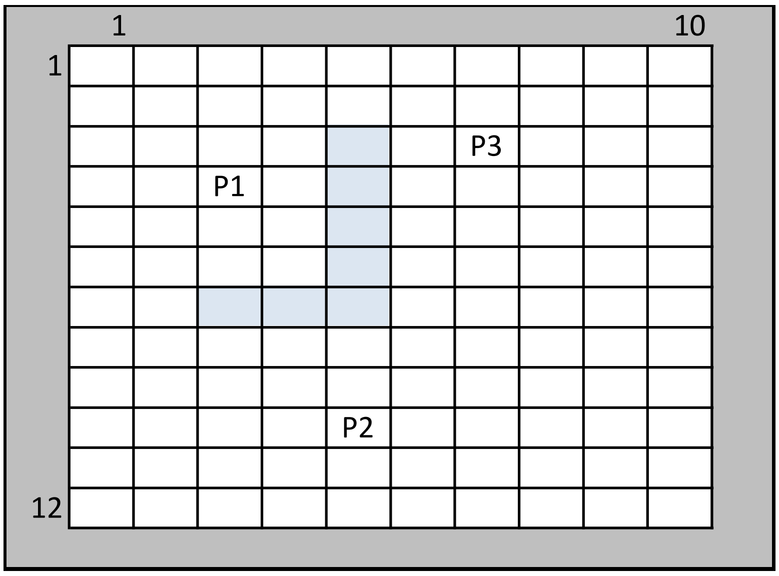


Table 3: Routing grid with three connection points P_1, P_2, and P_3 and an obstacle marked in blue.

The example shows the visualization of a chip with a grid size of 10 by 12. The blue cells are blocked cells, and the points P_1, P_2, and P_3 are connection points to be connected by a single net. Initially we will consider different algorithms for connecting P_1 and P_2, and then later we will extend the example to connect all of the three points.

5.1 Lee's algorithm

The Lee algorithm [6] is quite similar to a breadth-first search. It works by continuously applying layers as a propagating wave, filling the search space from the source in all directions. Because of this wave-like propagation it guarantees to always find the optimal solution if one exists [6]. In practice it works by visiting adjacent cells from the previously visited cells or from the source if at the start. It always finishes the previous wave before starting a new one.

The following pseudo code [6] describes the Lee algorithm:

```
Algorithm Lee-Router(B,s,t,P)
    input B, s, t
    output P
begin
    plist = s;
    nlist = $\phi$;
    temp = 1;
    path_exists = FALSE;
    while plist $\neq$ $\phi$;
        foreach vertex $v_i$ in plist do
            foreach vertex $v_j$ neighboring $v_i$ do
                if B[$v_j$] = UNBLOCKED then
                    L[$v_j$] = temp;
                    INSERT($v_j$,nlist);
                    if $v_j$ = t then
                        path_exists = TRUE;
                        exit while;
        temp = temp + 1;
        plist = nlist;
        nlist = $\phi$
    if path_exists = TRUE then RETRACE(L,P);
    else path does not exsist;
end
```

For the algorithm in its simplest form without blocked cells, the mode of operation is show in Table 4.

5	4	3	4	5	6	7	8		
4	3	2	3	4	5	6	7	8	
3	2	1	2	3	4	5	6	7	8
2	1	S/0	1	2	3	4	5	6	7
3	2	1	2	3	4	5	6	7	8
4	3	2	3	4	5	6	7	8	
5	4	3	4	5	6	7	8		
6	5	4	5	6	7	8			
7	6	5	6	7	8				
8	7	6	7	T/8					
	8	7	8						
		8							

Table 4: Routing grid based on Table 3 but without the obstacle. The route from P_1 to P_2 has been calculated with Lee's algorithm.

Initially the source 'S' is assigned the number of the initial wave which is zero. During the next iteration, all adjacent cells are visited and assigned the wave number one. Then all cells adjacent to cells visited in wave number 1 are visited and assigned the wave number 2. This continues until the sink 'T' is reached or the search space is exhausted.

When we add obstacles to the grid, then we simply need to check whether the adjacent cells are occupied or not before visiting them. An example of this can be seen in Table 5.

5	4	3	4	5	6	7	8	9	10
4	3	2	3	4	5	6	7	8	9
3	2	1	2	🟦	6	7	8	9	10
2	1	S/0	1	🟦	7	8	9	10	
3	2	1	2	🟦	8	9	10		
4	3	2	3	🟦	9	10			
5	4	🟦	🟦	🟦	10				
6	5	6	7	8	9	10			
7	6	7	8	9	10				
8	7	8	9	T/10					
9	8	9	10						
10	9	10							

Table 5: Routing grid based on Table 3, with route from P_1 to P_2 going around the obstacle. The route has been calculated with Lee's algorithm.

5.2 Hadlock's algorithm

The Hadlock algorithm [6] is based on the idea that the length of the optimal path between two vertices is equal to the Manhattan distance M(s,t) plus two times the number of cells it moves away from the target on this path d(P). The shortest path can then be calculated as having the distance M(s,t) + 2d(P). This means that if the detour in the path is minimized then so is the path itself.[6]

The following pseudo code [6] describes the Hadlock algorithm:

```
Algorithm HADLOCK-ROUTER(B,s,t,P)
    input B,s,t
    output: P
begin
    plist = s;
    nlist = $\phi$;
    detour = 0;
    path_exists = FALSE;
    while plist $\neq \phi$ do
        foreach vertex $v_i$ in plist do
            forall vertices $v_{j\ neighboring}$ $v_i$ do
                if B[$v_j$] = UNBLOCKED then
                    D[$v_j$] = DETOUR-NUMBER($v_j$);
                    INSERT($v_j$,nlist);
                    if $v_j$ = t then
                        path_exists = TRUE;
                        exit while;
        if nlist = $\phi$ then
            path_exist = FALSE;
            exit while;
        detour = MINIMUM_DETOUR(nlist);
        foreach vertex $v_k$ in nlist do
            if D[$v_k$] = detour then INSERT($v_k$,plist);
        DELETE(nlist,plist)
    if path_exists = TRUE then RETRACE(L,P);
    else path does not exist;
end;
```

Illustrated below, in Table 6, is the idea using the same example as previously.

		2	2						
	2	1	1	🟦					
2	1	S/0	0	🟦					
2	1	0	0	🟦					
2	1	0	0	🟦					
2	1	🟦	🟦	🟦					
2	1	1	1	1	2				
2	1	1	1	1	2				
2	1	1	1	T/1					
2	2	2	2						

Table 6: Routing grid based on Table 3, with route from P_1 to P_2 going around the obstacle. The route has been calculated with Hadlock's algorithm.

The propagation begins at the source 'S' with a detour value of 0. It now looks at the adjacent cells and assigns to them a detour value of 0, if the neighboring cells are closer to the sink 'T'. Otherwise the cells are assigned the detour value of 1. It then continues with the other cells that have the current minimum detour value of 0. When no more cells have the detour value of 0 then it continues with the cells that have a detour value of 1, assigning detour values of 1 and 2 respectively. The algorithm continues this until it reaches the sink 'T' or until the search space is exhausted. As with the Lee algorithm the Hadlock algorithm guarantees to find the shortest path if one exists.

After running either the Lee algorithm or the Hadlock algorithm, a simple back tracking algorithm can now retrace the shortest path from the sink to the source. This is done by starting at the sink 'T' and then repeatedly moving to the cell that holds the smallest value, until reaching the source 'S'.

5.3 Comparing the Lee and the Hadlock algorithm

The main advantage that the Hadlock algorithm has over the Lee algorithm is the average running time. The worst case running time for both the Lee and the Hadlock algorithm is $O(H \times W)$, but the running time of the Hadlock algorithm would often be much lower, as it always targets the sink when propagating.

Hadlock's algorithm functions by propagating towards a single sink, which means that it cannot have multiple different targets, and then simply connect to the closest. This is possible with the Lee algorithm as it propagates evenly in all directions, whereby it can simply terminate once it reaches the first sink.

5.4 Rectilinear Steiner Trees

The Lee and the Hadlock algorithms guarantee to find the optimal path between a source and a sink, but they are both unable to connect more than two points. This more complicated problem requires another approach.

Steiner Tree algorithms [6] aim to connect several points by a single net. A sub-problem of the Steiner Trees is Rectilinear Steiner Trees, which are trees with only rectilinear connections. These are connections with only vertical and/or horizontal paths. An example of a rectilinear Steiner Tree can be seen in Table 7.

Let us start by looking at the problem without blocked cells. Shown below in Table 7 is one of several possible rectilinear Steiner Tree solutions to the problem. The problem introduces a Steiner point here marked with an ‘S’. Later, we will see the significance of these Steiner points for the creation of intersections in the flow channel routing algorithm.

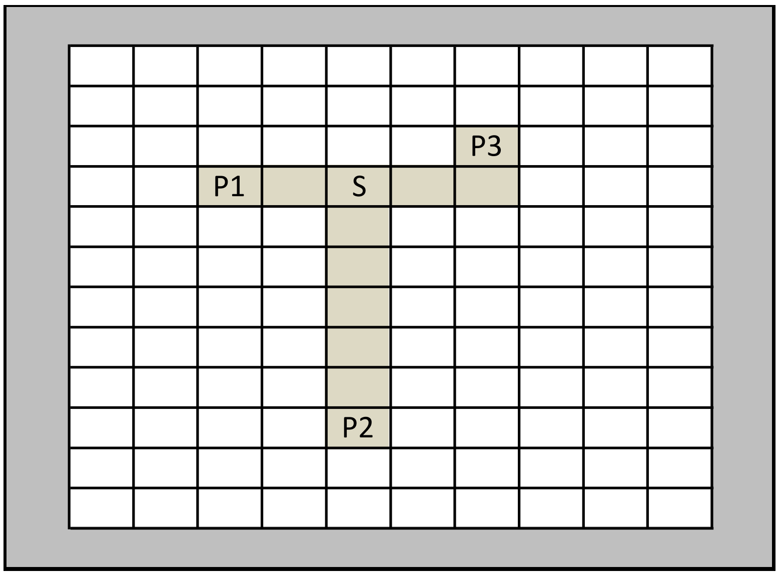


Table 7: Routing grid based on Table 3, with Rectilinear Steiner Tree routes connecting all three points P_1, P_2, and P_3.

5.5 Extending Lee and Hadlock

Returning to the path between P_1 and P_2 created above in Table 5 and Table 6, we now need to use the entire path between P_1 and P_2 as our starting points, in order to connect to the third point P_3.

Using all the points of the entire path as starting points we can now do the Lee wave-propagation just as we did previously. This will again guarantee that we find the shortest possible path to P_3 from the existing path between P_1 and P_2. Doing this Lee wave-propagation yields the solution shown below in Table 8.

4	3	3	4	5	6	7			
3	2	2	3	4	5	6	7		
2	1	1	2	🟦	6	T/7			
1	S/0	S/0	1	🟦	7				
1	S/0	1	2	🟦	6	7			
1	S/0	1	2	🟦	5	6	7		
1	S/0	🟦	🟦	🟦	4	5	6	7	
1	S/0	1	2	2	3	4	5	6	7
1	S/0	1	1	1	2	3	4	5	6
1	S/0	S/0	S/0	S/0	1	2	3	4	5
2	1	1	1	1	2	3	4	5	6
3	2	2	2	2	3	4	5	6	7

Table 8: Routing grid based on Table 3, with route to P_3 from a previously calculated route from P_1 to P_2. The route has been calculated with Lee's algorithm.

Unfortunately this way we are not guaranteed the overall shortest path between all points, since the order of which the points are connected matter. In fact if we had routed the other two points P_2 and P_3 first, then we would have attained a solution with a shorter path connecting all three points. This better solution, shown in Table 9, yields a path length of only 17 while the solution above in Table 8 has a length of 18.

	7	6	5	4	3	2	3	4	5
7	6	5	4	3	2	1	2	3	4
	7	6	5	🟦	1	S/0	1	2	3
		T/7	6	🟦	1	S/0	1	2	3
			7	🟦	1	S/0	1	2	3
	7			🟦	1	S/0	1	2	3
7	6	🟦	🟦	🟦	1	S/0	1	2	3
6	5	4	3	2	1	S/0	1	2	3
5	4	3	2	1	1	S/0	1	2	3
4	3	2	1	S/0	S/0	S/0	1	2	3
5	4	3	2	1	1	1	2	3	4
6	5	4	3	2	2	2	3	4	5

Table 9: Routing grid based on Table 3, with route going from a previously calculated route to P_1. The route has been calculated with the extended Lee algorithm.

Looking once more at the same problem but this time using the Hadlock propagation, we now get a different and non-optimal solution having a length of 20, as shown in Table 10. The problem here is that the small detour penalty introduced in row 2 prevents the algorithm from finding the optimal solution.

Therefore it does not guarantee to find the shortest path when routing from several source points.

1	1	1	1		1				
1	0	0	0	🟦	0	T/0			
1	S/0	S/0	0	🟦	0	0	1		
1	S/0	0	0	🟦	0	0	1		
1	S/0	0	0	🟦	0	0	1		
1	S/0	🟦	🟦	🟦	0	0	1		
1	S/0	0	0	0	0	0	1		
1	S/0	0	0	0	0	0	1		
1	S/0	S/0	S/0	S/0	0	0	1		
1	1	1	1	1	1	1			

Table 10: Routing grid based on Table 3, with route going from a previously calculated route to P_3. The route has been calculated with the extended Hadlock algorithm.

6 Algorithmic design

At the beginning of the project, we decided to define on a set of design rules and constraints, with regards to the algorithms and the data structures. These design rules are defined below:

- The biochip is defined as a 2-dimensional grid of evenly sized quadratic cells.
- The biochip will consist of a single flow layer, and a single control layer.
- The flow layer will contain both components, and flow channels.
- The control layer will contain control channels, and valves.
- Components are defined to be rectangular in shape, with a single input terminal and/or a single output terminal. If having two terminals these are always positioned on opposing sides of a component.
- Flow channels are defined to have a width equal to the width of a grid cell.
- Control channels are defined to have a width equal to one third the width of a flow channel.
- A valve is defined to have a size equal to the width of a control channel, which theoretically means that up to 9 valves can be fitted into a grid cell.
- Intersections are considered as being components, with either 3 or 4 valves regulating the flow through an equal number of flow lines.
- Control channels routed to components are only routed to the edge of the components. The routing from the edge of a component to the valves inside a component is considered a local problem and is therefore not considered by our algorithm.

- Control channels are not allowed to be routed over components. The reason for this is to avoid interference with the functionality and the internal routing of the components.
- Inlet/outlet ports for liquids entering the flow layer of the biochip may be positioned anywhere on the biochip area, and are treated as components.
- Inlet/outlet ports for high pressure air entering the control layer of the biochip are all positioned on the edges of the biochip.

All our work has been based on these design rules and constraints.

6.1 Placement

Since finding the optimal placement of components for a given design is an NP-complete problem an alternative approach is needed. We have decided to find inspiration in the nature inspired process of annealing metal [5].

The placement algorithm starts out by creating an initial random solution. This solution is limited by two factors being the user specified maximum size of the chip and the constraint of not allowing overlapping components. These two constraints are always enforced and they will be limiting all possible placements.

The algorithm relies on the following user defined variables and controls:

- ‘**Start temperature**’ – The initial temperature of the simulated annealing algorithm. The temperature determines the probability that an inferior offspring is selected for the next iteration.
- ‘**Alpha**’ – Rate of cooling of the simulated annealing algorithm.
- ‘**Iterations**’ – The total number of iterations of the simulated annealing algorithm.
- ‘**Rotation probability**’ – Allows the user to control the probability of whether a component is moved or rotated.
- ‘**Border width**’ – Determines the minimum distance between any pair of components.
- ‘**Inputs left / Outputs right**’ – Forces the components of type “Input” and “Output” to be placed in the leftmost quarter and rightmost quarter of the chip respectively.
- ‘**Move both directions**’ – Allow or disallow a component to be moved both vertically and horizontally within the same iteration.
- ‘**Standardized**’ – Forces the components to be placed in certain columns and rows. These columns and rows are spaced by the user specified variable ‘Std. width’.

Roughly the above described variables can be divided into two different categories. The ‘start temperature’, ‘alpha’ and the ‘iterations’ variables are related to the overall execution of the simulated annealing algorithm and the remaining variables and controls relate to the placement of the components on the chip.

The following pseudo code describes the algorithm:

```
$s = A\ random\ placement\ of\ all\ the\ components$
$t = 1$
$\alpha \in ]0:1[$
$T = Start\ temperature$
**while** $true$ **do**
    $s' = MoveOrRotateOneComponent(s)$
    $r = random\ number\ between\ 0\ and\ 1$
    **if** $f(s') \leq f(s) \vee e^{\frac{f(s)-f(s')}{\alpha^{t} \cdot T}} > r$ **then**
        $s = s'$
    **end if**
    $t = t + 1$
**end while**
```

As previously mentioned, the algorithm starts out by finding an initial solution that is used as an initial benchmark. After setting up the required parameters it continues with the first iteration of the algorithm.

At the beginning of each iteration an offspring *s'* is created based on the current solution *s*, by changing it slightly. In the pseudo code this is done by calling the function *MoveOrRotateOneComponent(S).* This function can perform two different modifications to the current solution in order to create the offspring. It can either rotate the component or it can move the component with respect to the value of the 'rotation probability' variable. Furthermore the exact execution of the function *MoveOrRotateOneComponent(S)* is highly dependent on several of the other variables and controls specified earlier.

As an extension we have made it possible for the user to lock the movement and rotation of selected components, if this is desired. By doing this a user can choose to let various components stay at fixed positions, while still letting the simulated annealing algorithm position the rest of the components.

Based on the solutions *s* and *s'*, we now decide which of them will be used in the next iteration. This is done by using the function *f(s)*, known as the fitness function. This fitness function evaluates the solution by calculating the Euclidian distances between all the connection points that must be connected.

Revisiting Figure 4, we can see that each of the lines and arrowheads represent a connection between two components. We make a distinction between the components' input and output terminals, with white arrowheads determining the direction of the connections. On the figure, component 1 has two outgoing connections, which will add to the value calculated by the fitness function. This increase will be equal to the sum of the distances from its outgoing terminal to the two ingoing terminals of the filter and the mixer.

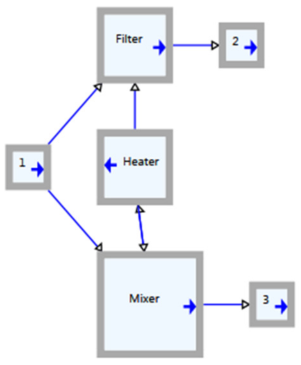


Figure 10: Manually done placement solution of the design shown on Figure 4.

The calculation of the fitness function for the example shown on Figure 10 would look like the following:

Dist(1.out, Filter.in) + Dist(1.out, Mixer.in) + Dist(Heater.out, Filter.in) + Dist(Heater.out, Mixer.in) + Dist(Mixer.out, Heater.in) + Dist(Mixer.out, 3.in) + Dist(Filter.out, 2.in)

Here *Dist(c1.out , c2.in)* calculates the Euclidian distance between c1's output terminal and c2's input terminal As defined under the rules, a component's input and output terminals are always on opposite sides of each other. Furthermore the output terminals are always located at the points pointed to by the solid blue arrows, just outside the components.

Based on the above definition of the fitness function, *f(s)*, we can now determine the Boolean value of the if-statement. If *f(s')* is smaller than or equal to *f(s)* then *s*' is used as the new base placement. Furthermore a small probability to continue with *s*' is introduced even though *s*' is not the better solution compared to *s*. This is done by checking if $e^{\frac{f(s)-f(s')}{\alpha^t \cdot T}} > r$, with r being a random number between 0 and 1. If the left hand side is larger than r, then we use the new solution *s*' in the next iteration. Otherwise it is discarded.

The final solution *s* is returned to the user after the specified number of iterations.

6.2 Flow channel routing

As previously mentioned, we need to identify components that require mandatory intersections. We have created a small custom algorithm for this specific problem.

6.2.1 Grouping terminals

If we again revisit the design from Figure 4, before routing the flow channels we need to determine which input and output terminals that must be connected. We will refer to these as a 'Terminal group'.

The following pseudo code describes the algorithm:

```
Create stack Next
Create list TerminalGroups
While non visited component c in Components
    Add c to Next
```

```
        Create a new TerminalGoup TG
        Mark c as visited
        **While** Next not empty
            c = Last element from Next
            Remove last element from Next
            Add c.out to TG if not in it allready
            **Foreach** component cTo in Components
                **if** c.out and cTo.in are connected **then**
                    Add cTo.in to TG if not in it allready
                    **Foreach** non visited component cFrom in Components
                        **if** cTo.In and cFrom.out are connected **then**
                            Add cFrom to Next if not visited
                            Mark cFrom as visited
                        **end if**
                    **end foreach**
                **end if**
            **end foreach**
        **end while**
        **If** TG contains more than one point **then**
            Add TG to TerminalGroups
        **end if**
    **end foreach**
```

To describe the process of finding terminal groups for the initial schematic design shown on Figure 4, we have created the table below. We will not go through the table step by step but instead define some simple rules.

- Every row needs to be visited once, which in the table below is equivalent to having a light red color.
- When the row gets a light red color then the outlet for the equivalent component row is added to 'Terminal Group'.
- For every '1' in the current row the column's inlet is added to the Terminal Group. This is marked with an *.
- If a column, marked with light blue, contains additional '1's, marked with a '+', then these rows are added to the 'Next' stack and the '1's are removed.
- A row can only be in the 'Next' stack once.
- Once the current row is done then the next is popped from the 'Next' stack.
- When the 'Next' stack is empty then the terminal group is defined and the next can be created.

The above steps will be repeated until all rows have been visited. An example of this can be seen in Table 11.

1

	To					
From	1	2	3	4	5	6
1	0	0	0	1	0	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	1	0	1
6	0	0	1	0	1	0

TG	Next
1.out	
TGList	

2

	To					
From	1	2	3	4	5	6
1	0	0	0	*	0	1
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	+	0	1
6	0	0	1	0	1	0

TG	Next
1.out	5
4.in	
TGList	

3

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	*
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	0	0	+
6	0	0	1	0	1	0

TG	Next
1.out	5
4.in	
6.in	
TGList	

4

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	0	0	0
6	0	0	1	0	1	0

TG	Next
1.out	
4.in	
6.in	
5.out	
TGList	

5

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	1	0	1
6	0	0	1	0	1	0

TG	Next
2.out	
TGList	
{1.out,4.in,6.in,5.out}	

6

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	1	0	1
6	0	0	1	0	1	0

TG	Next
3.out	
TGList	
{1.out,4.in,6.in,5.out}	

7

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	1	0	0	0	0
5	0	0	0	0	0	0
6	0	0	1	0	1	0

TG	Next
4.out	
TGList	
{1.out,4.in,6.in,5.out}	

8

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	*	0	0	0	0
5	0	0	0	0	0	0
6	0	0	1	0	1	0

TG	Next
4.out	
2.in	
TGList	
{1.out,4.in,6.in,5.out}	

9

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	1	0	1	0

TG	Next
6.out	
TGList	
{1.out,4.in,6.in,5.out}	
{4.out , 2.in}	

10

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	*	0	1	0

TG	Next
6.out	
3.in	
TGList	
{1.out,4.in,6.in,5.out}	
{4.out , 2.in}	

11

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	*	0

TG	Next
6.out	
3.in	
5.in	
TGList	
{1.out,4.in,6.in,5.out}	
{4.out , 2.in}	

12

	To					
From	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

TG	Next
TGList	
{1.out,4.in,6.in,5.out}	
{4.out , 2.in}	
{6.out , 3.in , 5.in}	

Table 11: Progression of the 'Grouping Terminals' algorithm.

After running the *GroupTerminals()* algorithm we now have a collection of nets, with each net consisting of a number of terminals/points that must be connected by the routing algorithm. This is accomplished by the *LeeSteiner()* algorithm, with the help of the *LeePropagation()* and *BackTrack()* algorithms.

6.2.2 Lee-Steiner algorithm

Our flow routing algorithm has been designed to take the following control variables:

- **T-cross Weight** – This variable controls how long a detour the algorithm will take to favor a connection to an already existing T-shaped intersection. This means that the generation of an intersection with 4 connections will be favored over two T-shaped intersections if the T-shaped intersections have a Manhattan distance between them that is smaller than the T-cross Weight. It should be noted though that we are using a weighted Manhattan distance, calculated with the use of a 2-dimensional map called 'CostMap', where each cell holds an individual contribution to the Manhattan distance calculation. The 'CostMap' and its functionality will be described in detail below.
- **Detour weight** – This variable controls how long a detour the algorithm will take in favor of creating non-mandatory crossings (see Figure 8). The detour cost is calculated with the use of the 'CostMap' mentioned in connection with the description of the T-cross Weight variable.
- **Buffer space** – This variable holds the size of the wanted free space around the components. The effect of setting this variable is that the flow channel routing algorithm will attempt to lay out the routes with at least a grid distance to the components equaling the value of this variable.

The flow routing algorithm uses three buffers for calculating routes that are all defined as 2-dimensional arrays. The first buffer is the 'RouteMap' where components and routed connections are stored. An example of this can be seen on Table 12.

				3	T	3			4	4	4		
					│		┌	─	T	4	4		
1	1	1		┌	┼	─	┘		4	4	4		
1	1	T	─	┘	│								
1	1	1			│			2	2	2			
					└	─	─	T	2	2			
								2	2	2			

Table 12: 'RouteMap' with three components with ID's '1', '2', and '3' and routed connections between them.

We have color coded the cells with regards to what they contain to make it easy to tell them apart. Components have been colored blue, previously routed control channels grey, and the newly routed control channel has been colored light red.

The 'RouteMap' buffer is divided into cells, each capable of holding either a certain component ID or a route ID. Before routing starts the components will be placed on the map at their individual positions and with their individual sizes. After routing has been completed the map will still hold the components but now also the flow channels connecting them. For the flow channel routing algorithm, the cells of the 'RouteMap' that contains routing definitions, holds explicit information about links to all the neighboring cells, instead of just a Boolean 'channel exists here'-variable. The reason for this is to avoid ambiguities with parallel channels, as they could be interpreted as being connected if the routing connections were not explicit.

The second buffer is the 'CostMap' which holds information about the cost of routing through the different cells of the 'RouteMap'. The 'CostMap' can be seen on Table 13.

2	2	2	3	🟦	🟦	🟦	3	3	🟦	🟦	🟦	3	2
3	3	3	3	3	3	3	3	3	🟦	🟦	🟦	3	2
🟦	🟦	🟦	3	2	2	2	2	3	🟦	🟦	🟦	3	2
🟦	🟦	🟦	3	2	1	2	3	3	3	3	3	3	2
🟦	🟦	🟦	3	2	1	2	3	🟦	🟦	🟦	3	2	2
3	3	3	3	2	1	2	3	🟦	🟦	🟦	3	2	1
2	2	2	2	2	1	2	3	🟦	🟦	🟦	3	2	1
1	1	1	1	1	1	2	3	3	3	3	3	2	1
1	1	1	1	1	1	2	2	2	2	2	2	2	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 13: 'CostMap' with values calculated on the basis of the components defined in the 'RouteMap' in Table 12.

The effect of using the 'CostMap' is that routes will try to stay away from 'expensive' cells. Then, if cells close to components are set to be more expensive than cells further away, the effect will be that routes will try to keep a certain minimum distance to the components. By doing this the risk of obstructing the flow channels of other components is reduced. If we take a look at Figure 11 we can see an illustration of this.

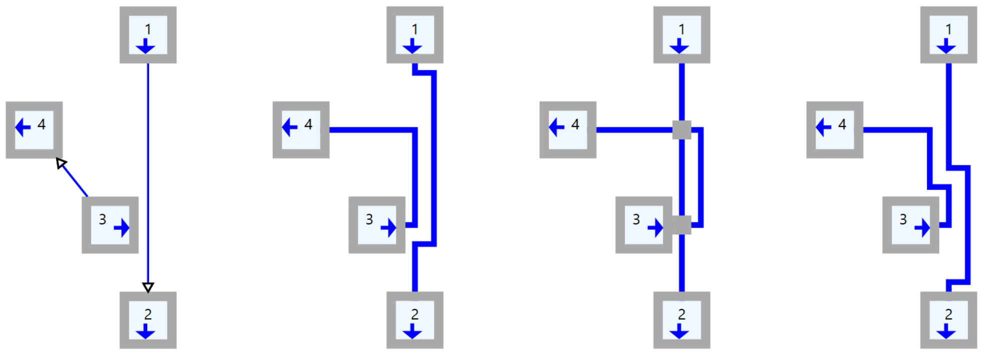


Figure 11: From left: Manually placed simple design[1]; automated routing[2] with Buffer Space=1; automated routing[3] with Buffer Space=1; automated routing[4] with Buffer Space=2.

Considering the manually placed design (1) in Figure 11, we have calculated three possible different routings.

On the first routing (2) in Figure 11, the order in which the flow channels have been routed is from component 3 to component 4 followed by a routing from component 2 to component 1 (backwards). This is fine and a good solution has been created.

For the next routing (3) in Figure 11, the order in which the flow channels have been routed is from component 1 to component 2 followed by a routing from component 3 to component 4. As can be seen, two non-mandatory intersections have been created. This is because the channel of the firstly routed net has effectively created an obstruction for the channel of the secondly routed net, because the first net has been laid out as close as is possible to the output port of component 3. Therefore, the only way to create a channel from component 3 to component 4 is to start by creating a non-mandatory intersection through the existing channel. Then because the Detour Weight variable is low, a second non-mandatory intersection is also created (the effect of setting the Detour Weight variable will be explained later). This is obviously not what we want, and thus it should be avoided.

For the last routing (4) in Figure 11, the order in which the flow channels have been routed is first from component 1 to component 2 followed by a routing from component 4 to component 3 (backwards). But as the Buffer Space variable has been increased to 2 the first routing will retain a minimum distance of 2 from component 3. This means that the second routing will have been given room for its channel, and the unwanted possibility of getting non-mandatory intersections with this design has been eliminated. It should be noted though that in many cases this problem should already have been eliminated during the placement process.

Returning to the description of the 'CostMap', the maximum value in each cell is determined by the variable named Buffer Space. Each cell effectively holds the bigger of either 1 or the Buffer Space minus the maximum distance in either the vertical or horizontal direction from itself to the border of the closest component, in either the vertical or horizontal direction, as defined in the 'RouteMap' (see Table 13). These values are calculated and stored on the 'CostMap' as components are placed on the 'RouteMap', by applying ever increasing rings with decreasing values around the placed components. In other words, the closest ring around a component gets the value Buffer Space, the next ring Buffer Space-1, and so on until applying a ring with the value of 1. However, any cell that holds a value that is greater than the ring value about to be set will be left unchanged.

The third buffer is the 'LeeMap'. This buffer holds the values of the waves that are the result of the Lee wave-propagation, as described in the section: Lee's algorithm, from the source points. It is in this buffer where far most of the computations are performed. The 'LeeMap' buffer can be seen on Table 14.

		19	17	🟦	S/5	🟦	17	20	🟦	🟦	🟦		
	20	17	14	11	8	11	14	17	🟦	🟦	🟦		
🟦	🟦	🟦	15	12	10	12	14	17	🟦	🟦	🟦		
🟦	🟦	🟦	16	13	11	13	16	19					
🟦	🟦	🟦	17	14	12	14	17	🟦	🟦	🟦			
		21	18	15	13	15	18	T	🟦	🟦			
		20	18	16	14	16	19	🟦	🟦	🟦			
	19	18	17	16	15	17	20						
		19	18	17	16	18	20						
			19	18	17	18	19						

Table 14: 'LeeMap' with calculated propagation values from 'S' to 'T', and with back tracking from 'T' to 'S' in red.

6.2.3 Lee wave-propagation

For simplicity, we will begin by looking at a slightly different example in order to show the effect of the T-cross weight variable, with regards to the Lee wave-propagation algorithm. As shown in Table 15, we try to connect 4 different components using a single net. The first 3 components have been connected by the brown net and we now use this as the base for the Lee wave-propagation. We have chosen to simplify this example by letting the cost of each cell be equal to 1. Furthermore, we will initially not favor the creation of a full intersection over a T-shaped intersection.

				4				4					
			4	3	2	S/1	2	3	4				
				2	2	S/1	2	3	4				
				S/1	S/1	S/1	2	3	4				
				2	2	S/1	2	3					
			4	3	2	S/1	2	3	T				
				4	2	S/1	2	3					
				4	2	S/1	2	3	4				
								4					

Table 15: LeeMap' with calculated propagation values, with T-cross Weight = 1, from the T-cross 'S' to 'T', and with back tracking from 'T' to 'S' in red.

By setting the T-cross weight variable to 1, we get the shortest possible route between the components and we end up with a solution that contains two Steiner points.

Now, let us look at the same problem again, but this time favor full intersections over T-cross intersections. We do this by allowing cells that contain a Steiner point to get a head start. The magnitude of the head start is determined by the T-cross weight. The Steiner point, shown in the example in Table 16 below, is given a head start of 4 cells, thus increasing the likelihood that it will be the first to reach the sink. The solution found in Table 16 has a slightly longer path that the solution found in Table 15, but it contains only one Steiner point. This will decrease the total number of required control valves by 2, which for some designs may be preferred.

								6					
					6	S/5	4	5	6				
				6	6	S/5	3	4	5	6			
				S/5	S/5	S/1	2	3	4	5	6		
				6	6	S/5	3	4					
					6	S/5	4	5	T				
					6	S/5	5	6					
					6	S/5	6						

Table 16: LeeMap' with calculated propagation values, with T-cross Weight = 5, from the T-cross 'S' to 'T', and with back tracking from 'T' to 'S' in red.

Having described the functionality of the T-cross weight variable, let us now re-consider the Lee wave-propagation information that is presented in Table 14. Here, the costs of the various cells are again defined by the 'CostMap'.

The flow channel routing of each connection starts with the generation of Lee wave-propagation data, starting from a collection of source points. In this example we will consider the single source point 'S'. Since this point is not a T-shaped intersection it is assigned the value contained in the T-cross Weight variable which in this example is set to 5. If it happened to be a T-shaped intersection it would have been assigned the value 1.

As waves are fully propagated in increasing order, starting with the smallest numbered wave the one to consider first is wave #5. From this cell it is only possible to propagate south, as the other three directions are blocked by component definitions. Adding the value 3, acquired from the equivalent southern cell in the 'CostMap', gives a 'LeeMap' cell value of 8. This value is stored in the southern cell in the 'LeeMap', since at the moment this is the first or, if not the first then the smallest, value in the cell. Also, this point is stored in a wave-buffer under the index for wave #8.

Having calculated new wave points (in this case just a single new point) from all the points in wave #5, the next and now the smallest wave #8 is considered. After Lee wave-propagation from wave #8 has been calculated the next wave #10 is considered, and so forth until the sink 'T' is reached. At this point the Lee wave-propagation algorithm stops and the back tracking algorithm takes over.

A specialized feature of our algorithm is its ability to create non-mandatory intersections. This feature is implemented by allowing the Lee wave-propagation algorithm to propagate into cells occupied by routes. When doing this the value to be added to the 'LeeMap' cell is not only the weight from the equivalent cell in the 'CostMap' but also the weight contained in the Detour Weight variable. As an example let us consider Table 17, where the propagation has been calculated with the Detour Weight variable set to 10:

23	21	19	17				33	36					
23	20	17	14	15	28	28	30	33	T				
			11	12	24	25	27	30					
		S/5	8	10	21	23	26	29	32	35			
			11	12	23	25	28						34
23	20	17	14	14	25	31	35					34	32
22	20	18	16	16	17	19	22				34	33	31
20	19	18	17	17	18	20	23	26	29	30	31	31	30
21	20	19	18	18	19	21	23	25	26	27	28	29	29
22	21	20	19	19	20	21	22	23	24	25	26	27	28

Table 17: 'LeeMap' containing obstructing flow channel in grey, with calculated propagation values from 'S' to 'T' using Detour Weight=10, and with back tracking from 'T' to 'S' in red.

As can be seen in Table 17, the propagation waves have been forced through the obstructing flow channel, shown in grey, using the weight of both the cell in the 'CostMap' and the Detour Weight variable, instead of just the weight from the 'CostMap'. It is worth noting that by doing this the propagation speed through the obstructing flow channel is effectively slowed down and controlled by the Detour Weight variable, thus facilitating the possibility of reaching the destination point 'T' via a detour instead.

In the example shown on Table 18, the Lee wave-propagation of the scenario of a detour beating the generation of a non-mandatory crossing is shown, as the Detour Weight variable is set to 40.

23	21	19	17					60				45	42
23	20	17	14	15	58	58	59	57	T			43	40
			11	12	54	55	56	54				41	38
		S/5	8	10	51	53	54	51	48	45	42	39	36
			11	12	53	55	57				39	36	34
23	20	17	14	14	55	61	65				37	34	32
22	20	18	16	16	17	19	22				34	33	31
20	19	18	17	17	18	20	23	26	29	30	31	31	30
21	20	19	18	18	19	21	23	25	26	27	28	29	29
22	21	20	19	19	20	21	22	23	24	25	26	27	28

Table 18: 'LeeMap' containing obstructing flow channel in grey, with calculated propagation values from 'S' to 'T' using Detour Weight=40, and with back tracking from 'T' to 'S' in red.

Note also the single empty cell on the top of the map. This is actually left empty by the algorithm, because the sink 'T' is reached while Lee wave-propagating from the points in wave #57, after which the Lee Wave-propagation algorithm stops. Yet another thing worth noting is the way that the light red flow route is laid out with a distance of three to the bottom right component. This is a result of using the weights of the 'CostMap', which in turn has been calculated on the basis of the Buffer Space variable.

The following pseudo code describes the Lee wave-propagation algorithm:

```
WaveNumber = 0
CurrentWave = list of points
do
    WaveNumber++
    CurrentWave = SourcePointsPool at WaveNumber
    foreach point p in CurrentWave
        Propagate north of p
        Propagate east of p
        Propagate south of p
        Propagate west of p
    end foreach
while destination not reached
```

The following pseudo code describes propagation to the north:

```
pn = point north of p
CMval = value of CostMap at position pn
if pn is within routing grid then
```

```
        if pn is destination point then
            signal destination reached
            return
        if no object placed at RouteMap at position pn then
            if no route placed at RouteMap at position pn then
                if LeeMap at position pn > LeeMap at position p + CMval OR unvisited then
                    LeeMap at position pn = LeeMap at position p + CMval
                    add point to SourcePointsPool at wave number = LeeMap at position pn
                else if LeeMap at position pn > LeeMap at position p + CMval + DETOURWEIGHT then
                    LeeMap at position pn = LeeMap at position p + CMval + DETOURWEIGHT
                    add point to SourcePointsPool at wave number = LeeMap at position pn
                end if
            end if
        end if
    end if
```

6.2.4 Wave buffer records

The 'SourcePointsPool' data structure, mentioned in the pseudo code, is a record of the waves that are to follow the current active wave, kept as an array of collections of points, where the collections of points of each wave is inserted in the array, at a position equivalent with the wave number. The data structure is shown in Table 19.

Wave number	Points
5	P_{51}
8	P_{81}
10	P_{100}
11	P_{111}, P_{112}, P_{113}
12	P_{121}, P_{122}, P_{123}
13	P_{131}, P_{132}, P_{133}
14	P_{141}, P_{142}, P_{143}, P_{144}, P_{145}, P_{146}

Table 19: Visualization of the 'SourcePointsPool' data structure containing Lee wave-propagation data from Table 14, Empty lines have been omitted.

After the Lee wave-propagation has been completed, a back tracking is performed. The actual route of the channel that is being routed is extracted at this point.

6.2.5 Back tracking algorithm

The back tracking algorithm works by initially evaluating the points adjacent to the sink 'T'. It then looks in all the four possible directions relative to 'T', to find the smallest neighboring 'LeeMap' cell value. The cell with the smallest value is then chosen and an initial direction pointing towards this cell is then calculated for the back tracking process to move in. If more than one 'LeeMap' cell holds a smallest value, one of these cells is chosen randomly.

The algorithm now runs until reaching the source point 'S', whilst continuously looking ahead, left, and right, respectively, relative to the current direction for the smallest 'LeeMap' cell value. This way the algorithm will create routes with as few bends as possible because the cell in the direction of movement will be favored. Examples of this are shown on the Table 14, Table 17, and Table 18, with the extracted routes shown in light red.

The back tracking algorithm creates a route as a coherent string of points, which is then returned to the main routing algorithm.

6.2.6 Flow channel routing main algorithm

The Lee wave-propagation algorithm and the back tracking algorithm are tied together to complete the flow channel routing algorithm. The main algorithm implements randomization in the order that the nets are routed, so as to introduce random variation in the solutions.

The following pseudo code describes the algorithm:

```
initiate RouteMap
initiate CostMap
foreach component c in components
    place c on RouteMap
    set cell cost values on CostMap relative to c
end for
TerminalGroups = GroupTerminals(connections)
foreach TerminalGroup net in TerminalGroups
    PropagationPoints = empty list of points
    move a random point from net to PropagationPoints
    foreach point T in net
        SourcePointsPool = Collection of empty lists of points
        foreach point p in PropagationPoints
            if RouteMap at position p is a T-cross channel then
                add p to SourcePointsPool at wave number = 1
            else
                add p to SourcePointsPool at wave number = T-cross Weight
            endif
        end foreach
        LeePropagation(SourcePointsPool)
        Add BackTrack(T) points to PropagationPoints
    end foreach
end foreach
```

Having completed all of the above, all the flow channels have now been routed, and all intersections, both mandatory and non-mandatory, have been defined. The chip is now ready for the next design steps.

6.3 Control channel routing

As mentioned in the problem description for the control channel routing, the problem of routing the control channels is very similar to the problem of routing nets on electronic circuit boards. Still, for routing the control channels we have re-used some of the techniques that we developed for routing the flow channels. We still make use of the 'RouteMap' and the 'LeeMap' buffers, but because the control channel routing algorithm implements rip-up of nets, instead of trying to make room around components for routing, the 'CostMap' is no longer relevant and therefore it has been removed. Also, the resolution of the buffers has been increased by a factor 3 in both the vertical and horizontal direction, because we defined the width of the control channels to be 1/3 the width of the flow channels.

The things that make control channel routing different from flow channel routing is that unrelated nets must not intersect each other; the number of nets to be routed is usually higher; and the combined length of the control channels is far greater than the combined length of the flow channels. Also, as a result of our design, the control channels must all be routed from the valves to the edge of the chip, because this is where we have placed all the high pressure inlet ports.

Our control channel routing algorithm has been designed to take a single control variable:

- '**PORT_SPACING**' – This variable controls the spacing between the connection points allocated on the edge of the chip area.

The connection points that are distributed evenly around the edge of the chip, have collectively been defined as a specialized single net, and stored in the 'RouteMap' buffer. It is special in the sense that the net is a collection of independent points, meaning not forming a line. An illustration of the connection points net can be seen in Table 20, where the points of the net are marked with the letter 'C'.

C			C			C			C			C	
		1	1	1									
		1	1	1									
C		1	1	1			3	3	3				
				a			3	3	3				
				a			3	3	3				
C	a	a	a	a									
				a	a	a	a	a	a	2	2	2	
										2	2	2	
C										2	2	2	

Table 20: Top left corner of 'RouteMap', with connection points denoted 'C', components with ID's 1, 2, and 3 and a control channel routing denoted 'a'.

6.3.1 Lee wave-propagation

The process of routing the control channels is quite similar to routing the flow channels, but with a few important changes to the Lee wave-propagation algorithm and the back tracking algorithm. As we have already described these two algorithms in detail, we will only highlight the differences.

One difference is that we now must be able to route from net to net and not just from net to point. This is easily done, as the Lee wave-propagation is already working with lists of points. The first thing to do is simply to initialize the 'SourcePointsPool' data structure with the starting points illustrated in Table 21, where the source points have been denoted 'S'. This is because we only route control lines to the edges of components, as previously described.

										7	7	T	
									7	6	6	6	7
								7	6	5	5	5	6
										4	4	4	5
						7				3	3	3	4
					7	6				2	2	2	3
				7	6	5	4	3	2	1	1	1	2
			7	6	5	4	3	2	1	S	S	S	1
			7	6	5	4	3	2	1	S		S	1
			7	6	5	4	3	2	1	S	S	S	1

Table 21: Top left corner of 'LeeMap', with Lee wave-propagation data from points on the edge of a component denoted 'S'.

In Table 21, and in all the following illustrated 'LeeMap' tables, we have embedded the information from the 'RouteMap' by color coding the different cells of the 'LeeMap'. Components have been colored blue, connection ports orange, previously routed control channels have been colored different shades of grey and blue, and the newly routed control channel has been colored light red.

From the source points denoted 'S' the first wave number 1 is propagated and the points have been stored in the 'LeeMap' buffer. From wave number 1 the next wave number 2 is propagated, then wave number 3, and so on, until at least 1 point of the destination net (denoted 'T' in Table 21) is reached. During the propagation process equivalent cells from the 'RouteMap' with component definitions are avoided. After reaching a destination point, the back tracking algorithm takes over and creates the new channel.

Illustrated on Table 22 is a newly created control channel routing, from one of the previously created control channels to component number 1. It should be noted that component number 2, as defined in Table 20, is now boxed in by other nets, and rip-up of one of these nets is now the only possibility for component number 2 to connect to an inlet port.

					8		6	5		2	1	S	1
		T	T	T	7	6	5	4	3	2	1	S	1
		T		T	7	6	5	4	3	2	1	S	1
		T	T	T	8	7				2	1	S	1
						8				2	1	S	1
					8	7				2	1	S	1
				8	7	6	5	4	3	2	1	S	1
					8	7	6	5	4				2
						8	7	6	5				3
							8	7	6				4

Table 22: Top left corner of 'LeeMap', with Lee wave-propagation data from points on a route denoted 'S'.

To intelligently decide what net to rip-up, we have made an alteration to the Lee wave-propagation algorithm. Like for the flow channel routing algorithm, we allow propagation into cells holding routes as defined in the equivalent cells on the 'RouteMap' buffer. However, instead of just slowing down propagation through these cells, we effectively pause the propagation, by storing these halted propagation points in a buffer. This buffer holds only points that are blocked by other channels, until the Lee wave-propagation stops. This is illustrated in Table 23, where the stored points are the ones that have Lee wave-propagation values written on top of the obstructing nets.

					3	2	1	1	1	2	3		
					2	1	S	S	S	1	2	3	
				3	2	1	S		S	1	2	3	
				3	2	1	S	S	S	1	2	3	
				4	3	2	1	1	1	2	3	4	
					4	3	2	2	2				

Table 23: Top left corner of 'LeeMap', with Lee wave-propagation data from points on the edge of a component denoted 'S'.

If the Lee wave-propagation stopped without reaching a destination point (which happened on Table 23), the halted points are then used for further propagation, until the algorithm stops again. This is done by re-numbering the stored waves, so as to match the current propagation wave number after which the Lee wave-propagation continues. If the Lee wave-propagation again stops without reaching a destination point, the new halted points are then used for further propagation, and so on, until a destination point is reached, which is what has happened in Table 24.

T			T			T	6	6	T			T	
						6	5	5	5	6			
					3	2	1	1	1	2	3		
T					2	1	S	S	S	1	2	3	
				3	2	1	S		S	1	2	3	
				3	2	1	S	S	S	1	2	3	
T				4	3	2	1	1	1	2	3	4	
					4	3	2	2	2				
							6	6	6				
T													

Table 24: Top left corner of 'LeeMap', with Lee wave-propagation data after invoking and re-numbering the buffer containing halted points.

It should be noted that, like for the flow channel routing, we route nets in a randomized order.

6.3.2 Back tracking algorithm with rip-up

Since a destination point has now been reached the back tracking algorithm starts back tracking from the destination point 'T', or if more than one destination point has been reached, one is chosen at random, from where the back tracking starts. The difference in the back tracking algorithm for the control channel routing algorithm, compared to the back tracking algorithm for the flow channel routing algorithm, is that no non-mandatory intersections are created, as this is not allowed. Instead, when the back tracking algorithm crosses over an obstructing control channel, the complete net that the obstructing control channel is a part of, is ripped up and put back in the collection of unrouted nets for re-routing at a later point. After ripping up the obstructing net the back tracking algorithm continues towards a source point, ripping up obstructing nets as they are encountered. The result of this back tracking can be seen on Table 25.

🟧			🟧			🟧	6	6	🟧			🟧	
		🟦	🟦	🟦		6	5	5	5	6			
		🟦	🟦	🟦	3	2	1	1	1	2	3		
🟧		🟦	🟦	🟦	2	1	🟦	🟦	🟦	1	2	3	
				3	2	1	🟦	🟦	🟦	1	2	3	
				3	2	1	🟦	🟦	🟦	1	2	3	
🟧	▨	▨	▨	4	3	2	1	1	1	2	3	4	
				▨	4	3	2	2	2	🟦	🟦	🟦	
							6	6	6	🟦	🟦	🟦	
🟧										🟦	🟦	🟦	

Table 25: Top left corner of 'LeeMap', with Lee wave-propagation data after back tracking and rip-up of obstructing net.

The algorithm will continue routing and ripping up nets until all nets have been successfully connected to an inlet port.

6.3.3 Poisson distributed random rip-up

The effect of using our intelligent rip-up scheme is that the nets that are chosen to be ripped up are the ones that are actually blocking other nets. However, this strategy may sometimes result in a 'ping pong' situation where nets are ripping each other up repeatedly, resulting in a deadlock situation. For making it possible for the algorithm to get out of these kinds of circular deadlocks we have, in addition to the intelligent rip-up chosen to add a random Poisson distributed rip-up scheme. This means that whenever an intelligent rip-up occur we additionally rip up a random number of nets determined by the Poisson distribution. The Poisson distribution is chosen due to its exponential decline with regards to point probabilities. We have chosen to implement it with an expected additional rip-up of 1 net using the following formula:

$$F(k) = \frac{\lambda^k \, e^{-\lambda}}{k!}, where \ \lambda = 1$$

Using the Poisson distribution of the formula above, the most likely number of additional nets to be randomly ripped up is 0 or 1 nets, which have an equal probability. The next most likely number is 2 nets, then 3 nets, and so forth.

Three examples of control channel routings can be seen on Figure 12, where the control channels have been drawn in red. All three control channel routings have been created on the basis of the second (right) flow channel routing in Figure 8.

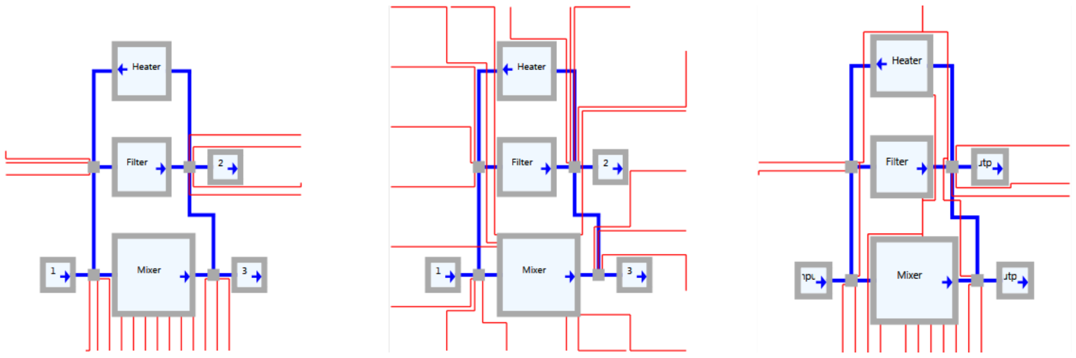


Figure 12: From Left: Control routing[1] with 'PORT_SPACING'=1, Control routing[2] with 'PORT_SPACING'=5, Control routing[3] with 'PORT_SPACING'=1 and manually defined control channel sharing.

As can be seen on the first control routing (1) on Figure 12, the valves controlling the flow through the intersections have all been routed to one of the high pressure inlet ports positioned on the edge of the biochip. As previously mentioned the spacing between these possible connection points is controlled by the 'PORT_SPACING' variable, which has been set to 1. For the second control routing (2) on Figure 12, the 'PORT_SPACING' variable has been set to 5, which causes the inlet ports to be much more evenly spaced.

The control synthesis process would normally have been done in between the flow channel routing and the control channel routing processes. But as we do not have an implementation of this, we have created a small example of control line sharing between valves, by manually altering the XML input file that is loaded by the control channel routing algorithm. The result of running the control channel routing algorithm on the basis of this manually altered XML input file, can be seen on the third control channel routing (3) on Figure 12.

7 Tool

The developed algorithms were integrated into a CAD tool which also provides the user with an intuitive way to create and modify schematic biochip architecture designs.

Based on an initial schematic design, the user can get support with the placement of various components, the routing of the flow channels and the routing of the control channels. We have divided the design process up into 4 different sequential steps that the user needs to go through, in order to create the final routing design of both layers of a continuous flow biochip. The 4 different sequential steps are:

- Schematic design (section 4.1)
- Placement (section 4.2)
- Flow channel routing (section 4.3)
- Control channel routing (section 4.6)

7.1 General

The Graphical User Interface (GUI) design for the tool, as can be seen on Figure 13, has been kept as clean as possible, and roughly, it can be divided into three different parts.

The first part is the controls on the left hand side. Here, we have implemented all the controls relating to the execution of the different algorithms. Here also are the controls for adding and removing components and connections to the schematic design and the placement.

The second part consists of six different tabs that contain either visual or tabular information about the different stages of the design process. These tabs will be discussed in the following sections.

The last part is the menu that the user can use to load and save files relevant for the steps in the design process. Also, it is possible to edit the component template library provided with the program.

We have incorporated an undo/redo design pattern, to enable the user to easily undo or redo any mistakes or changes done within the design and placement tab.

Furthermore, the user can easily navigate between previously routed flow and control solutions using either the menu or the left and right arrow keys, when in either the flow or control tab.

7.2 Design

The initial phase of the design process is to define the different components that are needed for the biochip application. After defining the relevant components the user should define the connections that are required between them. A screenshot of the design step can be seen on Figure 13.

Whenever a component is selected, all of its ingoing and outgoing connections are highlighted with a red color.

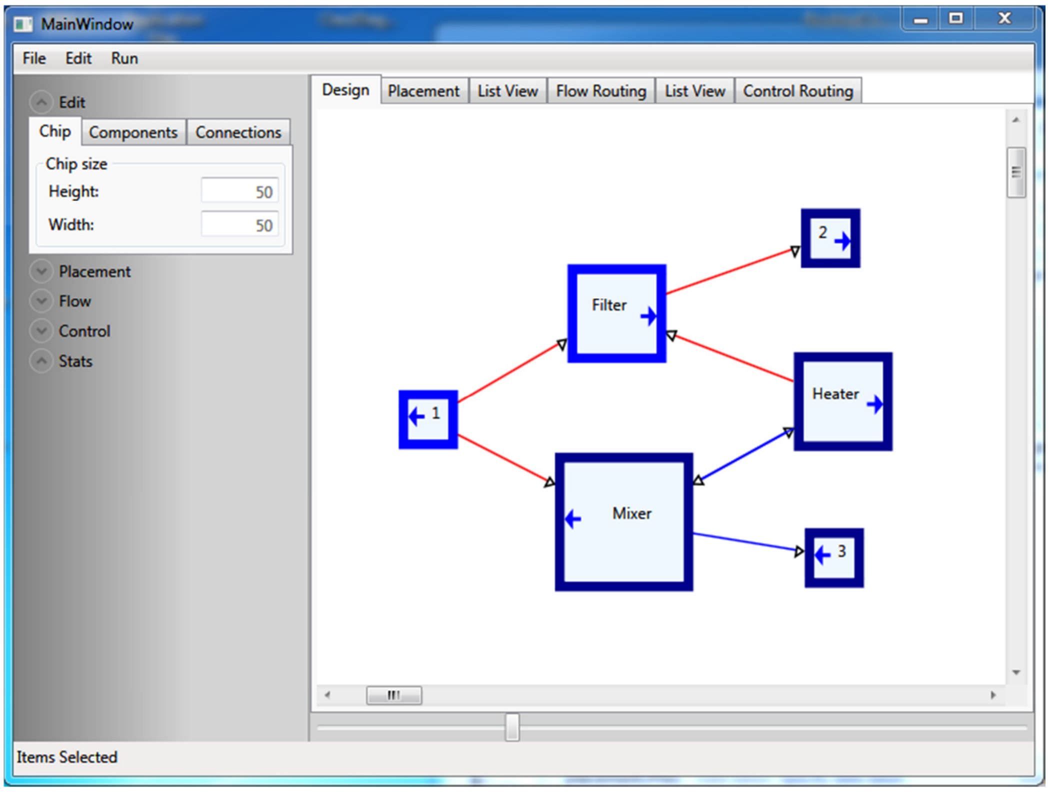


Figure 13: Design view of the custom biochip CAD tool.

The solid blue arrow inside the components always indicates the orientation of a component. The arrow points to the port, from which the liquid flows out of the component.

As part of the design process the user can choose to define new custom components, or change the definition of the component collection provided by the program. This is done through an XML-file containing the definitions. The user is free to add or modify components in the file. However, a restart of the program is required in order for the changes to take effect. The changes to the file only affect components added via the program and do not affect the files that are loaded.

As default the input data is defined as follows:

```
<Component>
	<Type>Input</Type>
	<StandardName>Input</StanddardName>
	<Size>
		<Height>3</Height>
		<Width>3</Width>
	</Size>
	<Valves>0</Valves>
</Component>
```

The user is provided with the following 5 components as standard components. They may be overwritten, but will otherwise remain as specified. The components are the following:

- Input
- Mixer

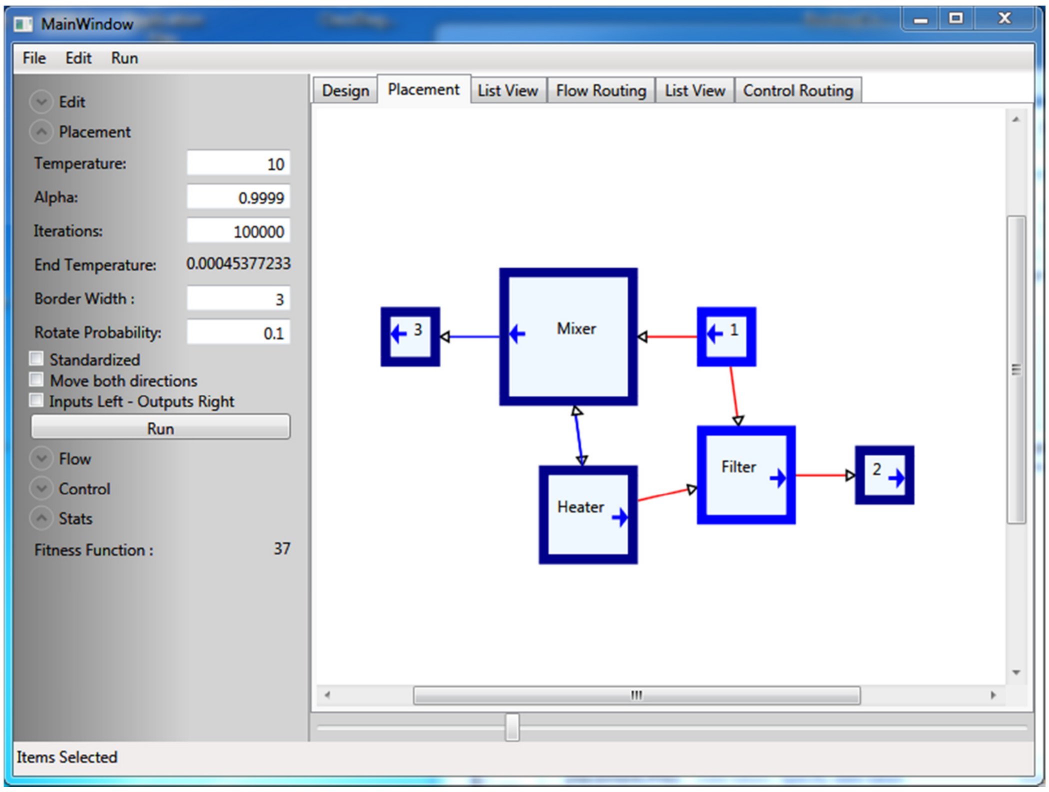
- Heater
- Filter
- Output

We have tried to make their sizes correct, in relation to each other, but with some degree of freedom for testing purposes. Inspiration related to size have been found in the literature [1].

7.3 Placement

The user may choose to manually place the different components or have our CAD tool aid the user with the placement process, using the Simulated Annealing algorithm presented earlier. The user can even specify placements of certain components and let the CAD tool find a suitable placement for the remaining components not locked in place by the user. Any locked components appear as grey instead of blue.

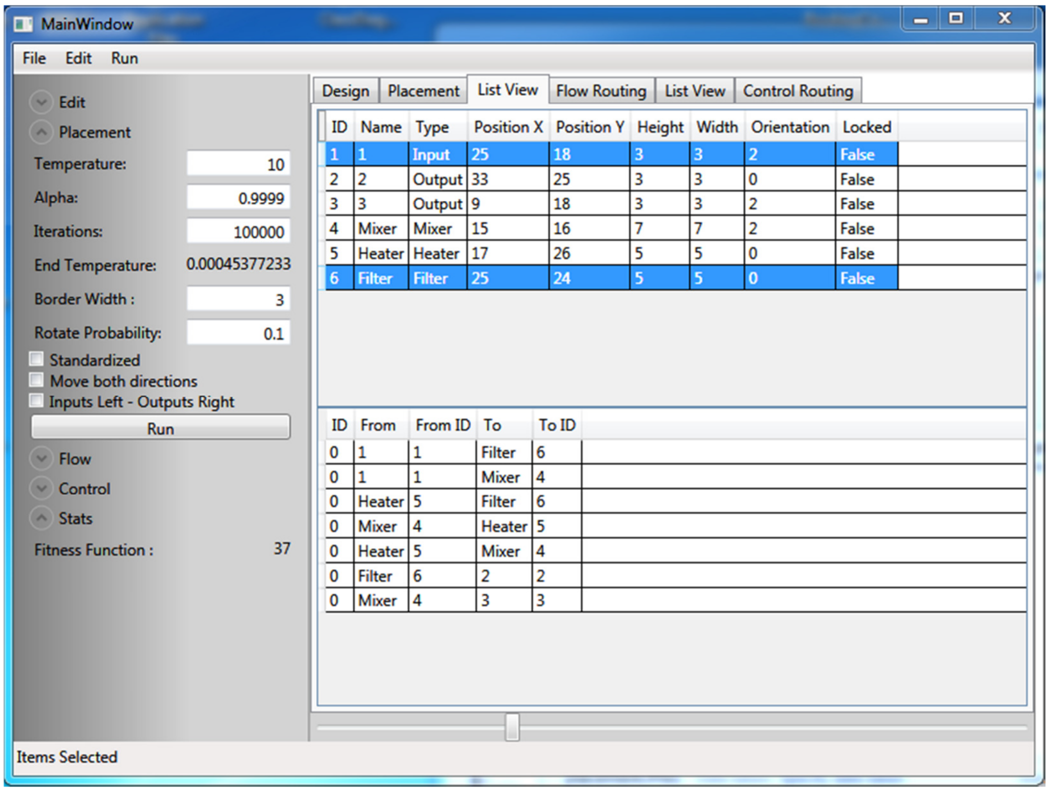
A screenshot of the placement step can be seen on Figure 14.



Temperature: 10
Alpha: 0.9999
Iterations: 100000
End Temperature: 0.00045377233
Border Width : 3
Rotate Probability: 0.1
Standardized
Move both directions
Inputs Left - Outputs Right
Fitness Function : 37
Items Selected

Figure 14: Placement view of the custom biochip CAD tool.

Once the user is satisfied with the placement of the components, the next step, which is routing of the flow layer, can commence. However, before this, the user is provided with information regarding the fitness function of the current solution. The fitness function is a cost-function based on the Euclidian distances between the input/output ports of the components that need to be connected. Also, a list view regarding the components and connections currently defined as part of the design can be seen in the List view tab, next to the placement tab, as shown on Figure 15.



Temperature: 10

Alpha: 0.9999

Iterations: 100000

End Temperature: 0.00045377233

Border Width : 3

Rotate Probability: 0.1

- [ ] Standardized
- [ ] Move both directions
- [ ] Inputs Left - Outputs Right

Fitness Function : 37

ID	Name	Type	Position X	Position Y	Height	Width	Orientation	Locked
1	1	Input	25	18	3	3	2	False
2	2	Output	33	25	3	3	0	False
3	3	Output	9	18	3	3	2	False
4	Mixer	Mixer	15	16	7	7	2	False
5	Heater	Heater	17	26	5	5	0	False
6	Filter	Filter	25	24	5	5	0	False

ID	From	From ID	To	To ID
0	1	1	Filter	6
0	1	1	Mixer	4
0	Heater	5	Filter	6
0	Mixer	4	Heater	5
0	Heater	5	Mixer	4
0	Filter	6	2	2
0	Mixer	4	3	3

Items Selected

Figure 15: List view of the placement of the custom biochip CAD tool.

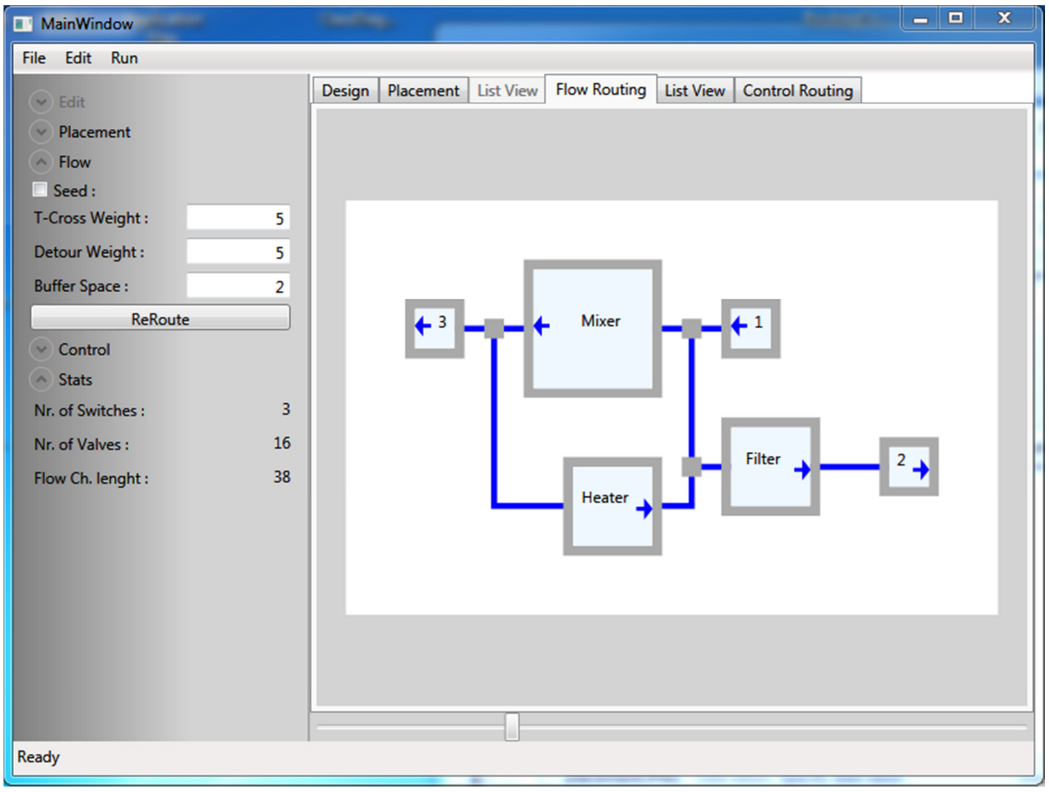
7.4 Flow Channel Routing

Firstly, it should be noted that the flow channel and the control channel routing algorithms have been developed and implemented as separate C++ programs. This was done in order to produce separate stand-alone programs, for each of the routers, which was a requirement for the project.

During the process of flow channel routing, the first of the external executable C++ programs is called to create the flow channels required. This routing step is done based on the actual placement, at the time of routing.

Communication between the design tool and the external routing programs is done by reading and writing XML-files. These files contain the relevant information about the components and connections. Additionally, user specified variables are passed on to the external program as runtime arguments.

The user is presented with the result of the flow channel routing once it is completed. The user cannot directly change this routing with the help of our program. That the components are locked are indicated by the grey color. Instead it is possible for the user to either manually alter the placement, or run the automated placement algorithm again, and then re-run the routing program. It is also possible to just re-run the routing program, and get a different flow channel routing solution. This can be done using either the same seed or a new seed for the randomization function.



Seed :
T-Cross Weight : 5
Detour Weight : 5
Buffer Space : 2
Nr. of Switches : 3
Nr. of Valves : 16
Flow Ch. lenght : 38

Figure 16: Flow channel routing view of the custom biochip CAD tool.

Once the user is content with the placement and the routing of the flow layer the control routing can commence, directly within the program.

In connection with the flow channel routing process, we provide some basic statistical information relating to the flow layer. The presented information shows the number of flow channel intersections, the number of control valves and the total length of the flow channels.

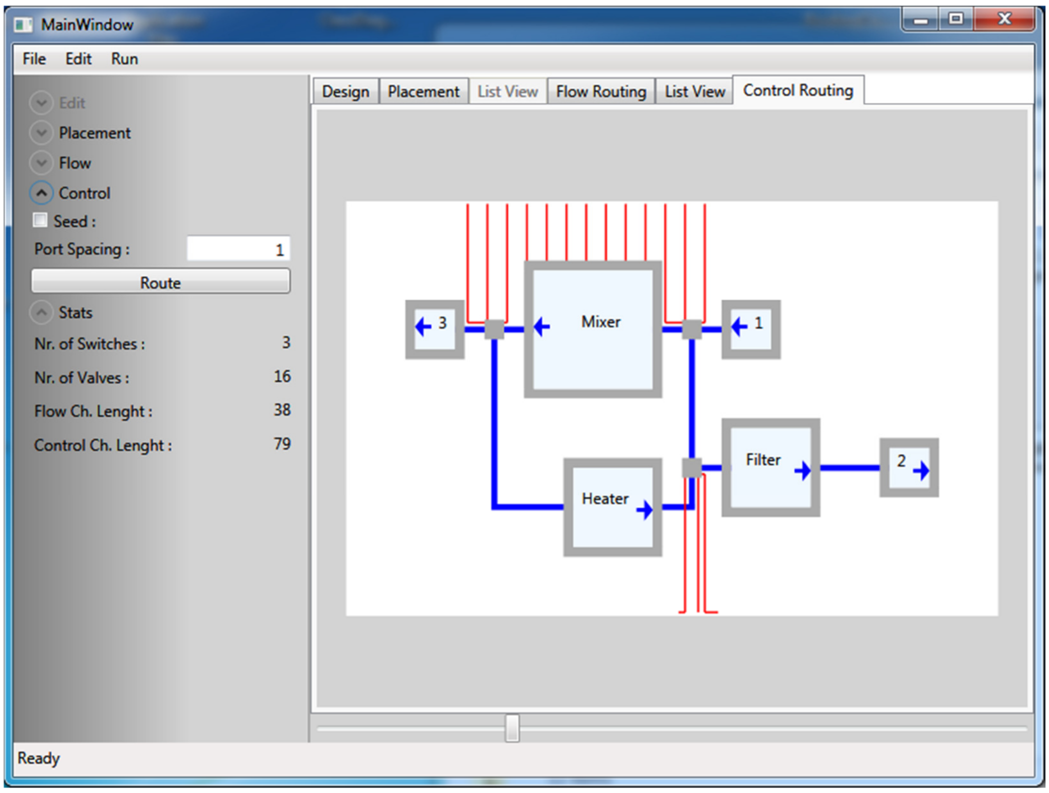
7.5 Application Mapping and Control Synthesis

The tool does not have application mapping and control synthesis as an integrated part. This means that if the user wishes to take advantage of the valve sharing in the control routing, then this should be done by saving the flow XML-file locally. This file should then be run through a third party application mapping and control synthesis application, such as the ones developed in [2], then through our control channel routing software, and then finally be loaded into our tool for visualization.

7.6 Control Channel Routing

The control channel routing part is the last part of the design process of the biochip that is covered by our program. A screenshot of this can be seen on Figure 17.

The control channel routing process is always based on the design shown in the Routing tab. This means that changes made in the placement tab do not affect the control channel routing, unless a new flow channel routing is done.



Seed :

Port Spacing : 1

Nr. of Switches : 3

Nr. of Valves : 16

Flow Ch. Lenght : 38

Control Ch. Lenght : 79

Figure 17: Control channel routing view of the custom biochip CAD tool.

It is not possible for the user to edit the placement of the components, or the connections between the components, at this point.

Also, for the control channel routing process, we provide some basic statistical information in relation to the control layer, like we do for the routing of the flow layer. The presented information shows the number of flow channel intersections, the number of control valves, the total length of the flow channels, and the total length of the control channels.

7.7 XML-Files

The XML-files are used to store schematic designs, placements, flow channel routings and control channel routings. We have used the same structure for all the four file-types, and the overall structure of our XML-files is discussed in the remainder of this section.

First, the XML-files specify the ID and the size of the biochip. Following this are the definitions of the components, the connections and the valves. This is the case of all the files, even when, for example, the paths of the connections are still unknown. In this case we simply leave the specific tags empty.

The overall schema is defined as follows:

```
</Architecture>
        <ID>"Integer"</ID>
        <Size>
                <Height>"Integer"</Height>
                <Width>"Integer"</Width>
        </Size>
        <ListOfArcComponents>
                //A number of components are defined
        </ListOfArcComponents>
        <ListOfArcConnectors>
                //A number of connections are defined
        </ListOfArcConnectors>
        <ListOfArcConnectedValves>
                //A number of Valves and their connections are defined
        </ListOfArcConnectedValves>
</Architecture>
```

The 'ListOfArcComponents'-tag signals the beginning of the definition of the components. Each Component is contained within an 'ArcComponentProperties'-tag and is defined by the following information:

- **ID –** The ID is used as an architecture-wide unique identification code of each of the components.
- **Name –** The name of a component is a user-defined string.
- **Type –** The type is a string defining the components intended use.
- **Orientation –** The orientation is an integer between 0-3 defining the orientation of the component.
    - A value of 0 indicates that the outgoing terminal points east.
    - A value of 1 indicates that the outgoing terminal points south.
    - A value of 2 indicates that the outgoing terminal points west.
    - A value of 3 indicates that the outgoing terminal points north.
- **Number of valves –** Defines the number of control valves needed to operate the component.
- **Size –** Information about the height and the width of the component.
- **Position –** The position of the upper left corner of the component on the biochip.

The 'ListOfArcComponents' structure is defined as follows:

```
<ListOfArcComponents>
        <ArcComponentProperties>
                <ID>"Integer"</ID>
                <Name>"String"</Name>
                <ArcComponentType>"String"</ArcComponentType>
                <Orientation>"Integer"</Orientation>
                <NumberOfValves>"Integer"</NumberOfValves>
                <Size>
                        <Height>"Integer"</Height>
                        <Width>"Integer"</Width>
                </Size>
                <Position>
                        <X>"Integer"</X>
                        <Y>"Integer"</Y>
                </Position>
        </ArcComponentProperties>
        ...
</ListOfArcComponents>
```

The next part of the XML-file, which is the ‘ListOfArcConnectors’, describes the connections defined during the design face by the user. Furthermore the specific paths between components will be show here if the flow channel routing has been done.

The ‘ListOfArcConnectors’ part, holds a number of ‘ArcConnectorProperties’ which in turn are defined by the following information:

- **Component –** Contains the definition of two different components. The first component specified is a ‘from’ component and the second is a ‘to’ component.
    - **ID –** The ID is the same unique identification code that is described above.
    - **Name –** This is the name of the component with the ID above.
    - **ConnectionPoint –** Contains information about the position of the relevant connection point, either an input or an output point.
- **LinePoints –** Corner points of the channel connecting the two connection points. A point is defined for each 90° turn that the path makes. The points are only defined once the flow routing has been done.

The ‘ListOfArcConnectors’ structure is defined as follows:

```
<ListOfArcConnectors>
        <ArcConnectorProperties>
                <Component>
                        <ID>"Integer"</ID>
                        <Name>"String"</Name>
                        <ConnectionPoint>
                                <X>"Integer"</X>
                                <Y>"Integer"</Y>
                        </ConnectionPoint>
                </Component>
                <Component>
                        <ID>"Integer"</ID>
                        <Name>"String"</Name>
                        <ConnectionPoint>
                                <X>"Integer"</X>
                                <Y>"Integer"</Y>
                        </ConnectionPoint>
                </Component>
                <LinePoints>
                        </Point>
                                <X>"Integer"</X>
                                <Y>"Integer"</Y>
                        </Point>
                        ...
                </LinePoints>
        </ArcConnectorProperties>
        ...
</ListOfArcConnectors>
```

The next section, which is the ‘ListOfArcConnectedValves’, defines the valves that are required by a given application. These are defined as a number of connected valves that shares a control line.

- **ID –** The ID is a unique identifier for the group of valves sharing a control line.
- **Valve –** Information about a single valve within the valve group. There can be any number of valves defined here.

    - **ValveID –** A unique identifier for the given valve.
    - **Object –** The name of the component for which this valve is needed.
    - **ObjectID –** The unique identifier for the specific component defined previously.
- **LineSegments –** Information about the path that connects the valves and an high pressure air inlet port.
    - **LineSegment –** Two points that define a section of the control channel needed to connect the valves with an inlet port. The size of these number should be divided by three when used.

The ‘ListOfArcConnectedValves’ structure is defined as follows:

```
<ListOfArcConnectedValves>
        <ConnectedValves>
                <ID>"Integer"</ID>
                <Valve>
                        <ValveID>"Integer"</ValveID>
                        <Object>"String"</Object>
                        <ObjectID>"Integer"</ObjectID>
                </Valve>
                ...
                <Valve>
                        <LineSegments>
                                <LineSegment>
                                        <Point>
                                              <X>"Integer"</X>
                                              <Y>"Integer"</Y>
                                        </Point>
                                        <Point>
                                              <X>"Integer"</X>
                                              <Y>"Integer"</Y>
                                        </Point>
                                </LineSegment>
                                ...
                        </LineSegments>
                </Valve>
        </ConnectedValves>
        ...
</ListOfArcConnectedValves>
```

7.8 Test

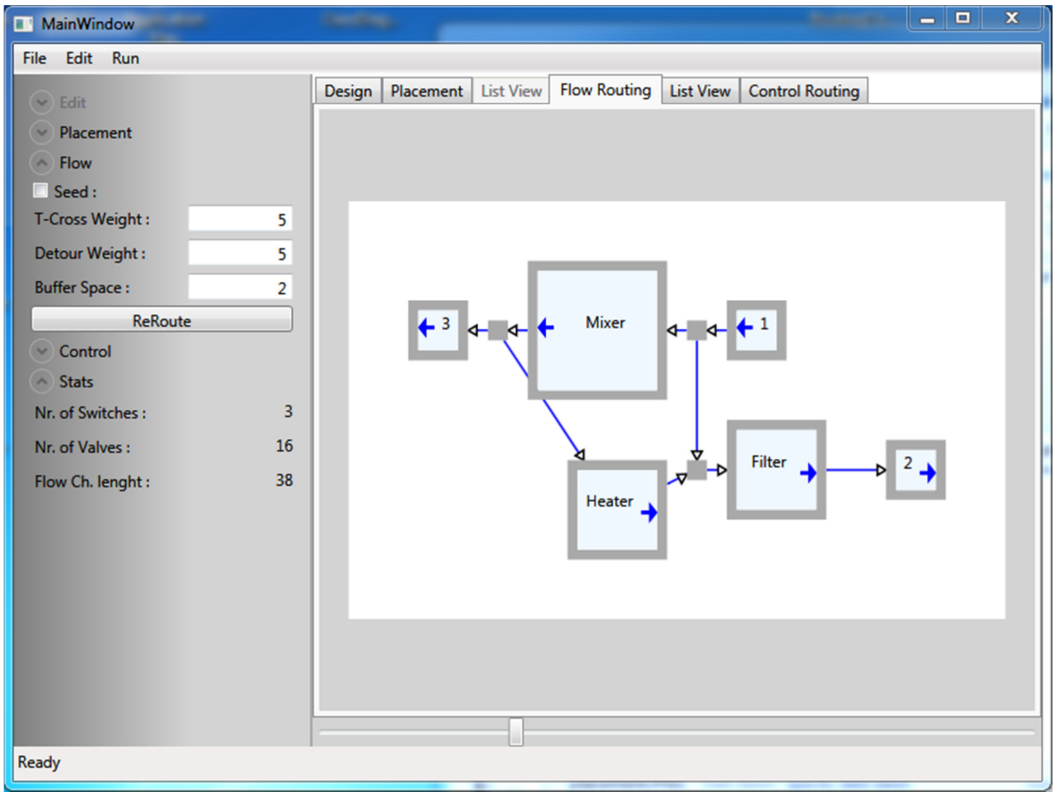
We have been using several different test and validation techniques in order to ensure the correctness of our algorithms.

During the implementation of the algorithms we have been using the debugger within visual studio, in order to ensure that the various maps/buffers have been used correctly.

Furthermore we have developed an integrated design and visualization tool, in order to enhance the inspection quality and speed of both designing and checking the placement and routing processes. We did this because checking the XML-files manually, very quickly became impossible, as the designs grew larger.

Also, as we wanted to create even larger designs with even more components and connections, we absolutely needed a streamlined tool for creating errorless XML-files. We therefore took the decision to include the design process into the tool.

We have created a debug mode, shown on Figure 18, in the flow part of the program. This allows us to see the connections between components directly instead of their paths. During the design and implementation phase we have enabled movement of the already routed components, to visually inspect and ensure correctness of the automated routing and intersection creation.



Seed :
T-Cross Weight : 5
Detour Weight : 5
Buffer Space : 2
Nr. of Switches : 3
Nr. of Valves : 16
Flow Ch. lenght : 38

Figure 18: Debug mode of the flow channel routing phase of the custom biochip CAD tool.

For testing the algorithms, we have produced and verified several designs, of which a few of them are shown on the front page and in the figures in this thesis. During this process, the setting of the variables has been tested and also a great amount of different designs have been placed, and routed.

7.9 Performance

Our placement algorithm is highly dependent on the size of the various components used. This is due to the fact that whenever a component is placed, the algorithm needs to check if the placement is valid and non-overlapping. Furthermore, the size of the chip influences the running time. This is due to the fact that for too small a chip the number of failed placement attempts will increase. On the other hand if the chip becomes too large, then more component moves will end up being bad placements, which will be rejected. This means that the user needs to experiment with the different controls and variables and the number of components to chip size ratio. Naturally the number of iterations requested by the user directly influences the running time of the algorithm.

The flow routing algorithm is highly dependent on the placement of the components and the chip size. As the chip size increases then so does the worst case running time. The running time is furthermore dependent on the number of points that needs to be connected.

We have experimented with several different biochip sizes ranging from 5 – 25 components and about 50 connections. These designs have all resulted in running times of less than a few seconds.

The routing of the control layer is dependent on the number of valves to be connected to high pressure air inlet ports. Also, the density of the valves in specific areas and the number of available inlet ports is very important. Lastly, because we do not allow control channels to pass over components, the running time can increase significantly. Therefore, the average running time of the algorithm is highly dependent on the placement and the routing of the flow layer previously created.

Again, we have experimented with several different biochip sizes. But because of the above mentioned factors, the running time may vary from a few seconds up to indefinitely, because some of the designs may be impossible to complete. In these situations it is up to the designer to change the placement or the routing of the flow channels, in order to successfully route the control channels. As a pointer, for the solution on the front page, the routing of the control channels took around 2 seconds.

8 Possible extensions

The process of creating this application has inspired us to come up with a few extensions to the algorithms and the data structures that could have been implemented.

One possible extension that might be worth considering is to allow the control routing algorithm to create routes using two layers. This could be done using a specialized algorithm that determines with control layer the valves should be placed, i.e. if it is a push-up or push-down valve. A naïve approach could be to simply route all nets that are ripped-up on the control layer on the opposite side, thus leaving room where needed.

Another possible extension that could have been considered, for the flow channel routing algorithm, is the rip-up or the shove functionality.

Yet another possible, but rather more radical, extension or alteration is to define a biochip grid consisting of homogenous hexagonal cells, instead of square cells. The advantage of defining a grid like this would be that more direct routings could be made. Also, to accomplish this, our currently implemented algorithms would only require minor changes to function with a hexagonal grid. The disadvantage of implementing a hexagonal grid would be a slight increase in algorithmic complexity.

9 Conclusion

The overall goal of this project has been to identify and implement different algorithms, for aiding the process of designing continuous flow-based biochips. As part of this process we have looked at the Lee and the Hadlock algorithms, as possible candidates for the solution of the routing problem of microfluidic Very Large Scale Integration. Our study and initial implementation lead to important insights into the problem.

As a result of these new insights we chose to extend the algorithms, so as to make it possible to create Rectilinear Steiner Trees.

Our study also revealed two weaknesses related to extending the Hadlock algorithm. Firstly, the Hadlock algorithm requires a single specific target in order to determine the detour values. This means that routing to the closest point out of several points is simply not possible. Secondly, the Hadlock algorithm does not guarantee to find the shortest path between a terminal and several starting points. Therefore, in our opinion, these two weaknesses make Hadlock inferior to the Lee algorithm.

Besides the routing algorithms, we have created an algorithm for identifying and grouping input and output terminals, that needs to be connected as a result of the design. This algorithm creates so called mandatory intersections between flow channels that must be merged, when two or more flow channels connect to a single terminal.

As we have seen, routed nets may end up blocking the channels of other nets that must also be routed. In the literature that we have read, we have not found any specific descriptions of how obstructing nets should be identified for rip-up and re-routing, apart from a simple strategy of randomly selecting nets. Because of this we have come up with and developed an intelligent rip-up algorithm. Furthermore, in order to prevent possible deadlocks, we implemented a Poisson distributed random rip-up scheme that works alongside our intelligent rip-up algorithm. This enables the algorithm to get out of situations where it might otherwise end up being stuck.

Additionally we have implemented a CAD tool that aims to support the user through the entire design process, from the initial schematic design through placement of components, routing of flow channels, and routing of control channels.

Lastly, we have implemented a simple solution to the placement problem, and we have integrated it into the CAD tool. The Simulated Annealing-based heuristic is inspired by the process of annealing metal and solves the placement problem through randomization, which has proven to be quite effective.

10 Bibliography

[1] J. P. Urbanski, W. Thies, C. Rhodes, S. Amarasinghe, and T. Thorsen, "Digital microfluidics using soft lithography," *Lab. Chip*, vol. 6, no. 1, p. 96, 2006.

[2] W. H. Minhass, P. Pop, and J. Madsen, "System-Level Modeling and Synthesis Techniques for Flow-Based Microfluidic Very Large Scale Integration Biochips," Technical University of Denmark, Department of Information Technology.

[3] "Microfluidics - Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/wiki/Microfluidics. [Accessed: 26-Jun-2013].

[4] "Stephen Quake's Group @ Stanford University." [Online]. Available: http://thebigone.stanford.edu/. [Accessed: 01-Jul-2013].

[5] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, "Global and Detailed Placement," in *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Dordrecht: Springer Netherlands, 2011, pp. 93–128.

[6] N. Sherwani, "Global Routing," in *Algorithms for VLSI Physical Design Automation*, Springer, 1995, pp. 223–266.

[7] W. A. Dees Jr and P. G. Karger, "Automated rip-up and reroute techniques," in *Proceedings of the 19th Design Automation Conference*, 1982, pp. 432–439.

11 Appendix A

Below is a sample XML-file of the design shown on Figure 4.

```xml
<Architecture>
    <ID>1</ID>
    <Size>
        <Height>21</Height>
        <Width>33</Width>
    </Size>
    <ListOfArcComponents>
        <ArcComponentProperties>
            <ID>1</ID>
            <Name>1</Name>
            <ArcComponentType>Input</ArcComponentType>
            <Orientation>2</Orientation>
            <NumberOfValves>0</NumberOfValves>
            <Size>
                <Height>3</Height>
                <Width>3</Width>
            </Size>
            <Position>
                <X>19</X>
                <Y>5</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>2</ID>
            <Name>2</Name>
            <ArcComponentType>Output</ArcComponentType>
            <Orientation>0</Orientation>
            <NumberOfValves>0</NumberOfValves>
            <Size>
                <Height>3</Height>
                <Width>3</Width>
            </Size>
            <Position>
                <X>27</X>
                <Y>12</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>3</ID>
            <Name>3</Name>
            <ArcComponentType>Output</ArcComponentType>
            <Orientation>2</Orientation>
            <NumberOfValves>0</NumberOfValves>
            <Size>
                <Height>3</Height>
                <Width>3</Width>
            </Size>
            <Position>
                <X>3</X>
                <Y>5</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>4</ID>
            <Name>Mixer</Name>
            <ArcComponentType>Mixer</ArcComponentType>
            <Orientation>2</Orientation>
            <NumberOfValves>7</NumberOfValves>
            <Size>
                <Height>7</Height>
                <Width>7</Width>
            </Size>
            <Position>
                <X>9</X>
                <Y>3</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>5</ID>
            <Name>Heater</Name>
            <ArcComponentType>Heater</ArcComponentType>
            <Orientation>0</Orientation>
            <NumberOfValves>0</NumberOfValves>
            <Size>
                <Height>5</Height>
                <Width>5</Width>
            </Size>
            <Position>
                <X>11</X>
                <Y>13</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>6</ID>
            <Name>Filter</Name>
            <ArcComponentType>Filter</ArcComponentType>
            <Orientation>0</Orientation>
            <NumberOfValves>0</NumberOfValves>
            <Size>
                <Height>5</Height>
                <Width>5</Width>
            </Size>
            <Position>
                <X>19</X>
                <Y>11</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>7</ID>
            <Name>S7</Name>
            <ArcComponentType>switchESW</ArcComponentType>
            <Orientation>1</Orientation>
            <NumberOfValves>3</NumberOfValves>
            <Size>
                <Height>1</Height>
                <Width>1</Width>
            </Size>
            <Position>
                <X>7</X>
                <Y>6</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>8</ID>
            <Name>S8</Name>
            <ArcComponentType>switchESW</ArcComponentType>
            <Orientation>1</Orientation>
            <NumberOfValves>3</NumberOfValves>
            <Size>
                <Height>1</Height>
                <Width>1</Width>
            </Size>
            <Position>
                <X>17</X>
                <Y>6</Y>
            </Position>
        </ArcComponentProperties>
        <ArcComponentProperties>
            <ID>9</ID>
            <Name>S9</Name>
            <ArcComponentType>switchNES</ArcComponentType>
            <Orientation>1</Orientation>
            <NumberOfValves>3</NumberOfValves>
            <Size>
                <Height>1</Height>
                <Width>1</Width>
            </Size>
            <Position>
                <X>17</X>
                <Y>13</Y>
            </Position>
        </ArcComponentProperties>
    </ListOfArcComponents>
```

```
<ListOfArcConnectors>
        <ArcConnectorProperties>
                <Component>
                        <ID>1</ID>
                        <Name>1</Name>
                        <ConnectionPoint>
                                <X>19</X>
                                <Y>6</Y>
                        </ConnectionPoint>
                </Component>
                <Component>
                        <ID>8</ID>
                        <Name>S8</Name>
                        <ConnectionPoint>
                                <X>17</X>
                                <Y>6</Y>
                        </ConnectionPoint>
                </Component>
                <LinePoints>
                </LinePoints>
        </ArcConnectorProperties>
        <ArcConnectorProperties>
                <Component>
                        <ID>4</ID>
                        <Name>Mixer</Name>
                        <ConnectionPoint>
                                <X>9</X>
                                <Y>6</Y>
                        </ConnectionPoint>
                </Component>
                <Component>
                        <ID>7</ID>
                        <Name>S7</Name>
                        <ConnectionPoint>
                                <X>7</X>
                                <Y>6</Y>
                        </ConnectionPoint>
                </Component>
                <LinePoints>
                </LinePoints>
        </ArcConnectorProperties>
        <ArcConnectorProperties>
                <Component>
                        <ID>5</ID>
                        <Name>Heater</Name>
                        <ConnectionPoint>
                                <X>15</X>
                                <Y>15</Y>
                        </ConnectionPoint>
                </Component>
                <Component>
                        <ID>9</ID>
                        <Name>S9</Name>
                        <ConnectionPoint>
                                <X>17</X>
                                <Y>13</Y>
                        </ConnectionPoint>
                </Component>
                <LinePoints>
                        <Point>
                                <X>17</X>
                                <Y>15</Y>
                        </Point>
                </LinePoints>
        </ArcConnectorProperties>
        <ArcConnectorProperties>
                <Component>
                        <ID>6</ID>
                        <Name>Filter</Name>
                        <ConnectionPoint>
                                <X>23</X>
                                <Y>13</Y>
                        </ConnectionPoint>
                </Component>
        <Component>
                <ID>2</ID>
                <Name>2</Name>
                <ConnectionPoint>
                        <X>27</X>
                        <Y>13</Y>
                </ConnectionPoint>
        </Component>
        <LinePoints>
        </LinePoints>
</ArcConnectorProperties>
<ArcConnectorProperties>
        <Component>
                <ID>8</ID>
                <Name>S8</Name>
                <ConnectionPoint>
                        <X>17</X>
                        <Y>6</Y>
                </ConnectionPoint>
        </Component>
        <Component>
                <ID>9</ID>
                <Name>S9</Name>
                <ConnectionPoint>
                        <X>17</X>
                        <Y>13</Y>
                </ConnectionPoint>
        </Component>
        <LinePoints>
        </LinePoints>
</ArcConnectorProperties>
<ArcConnectorProperties>
        <Component>
                <ID>8</ID>
                <Name>S8</Name>
                <ConnectionPoint>
                        <X>17</X>
                        <Y>6</Y>
                </ConnectionPoint>
        </Component>
        <Component>
                <ID>4</ID>
                <Name>Mixer</Name>
                <ConnectionPoint>
                        <X>15</X>
                        <Y>6</Y>
                </ConnectionPoint>
        </Component>
        <LinePoints>
        </LinePoints>
</ArcConnectorProperties>
<ArcConnectorProperties>
        <Component>
                <ID>7</ID>
                <Name>S7</Name>
                <ConnectionPoint>
                        <X>7</X>
                        <Y>6</Y>
                </ConnectionPoint>
        </Component>
        <Component>
                <ID>5</ID>
                <Name>Heater</Name>
                <ConnectionPoint>
                        <X>11</X>
                        <Y>15</Y>
                </ConnectionPoint>
        </Component>
        <LinePoints>
                <Point>
                        <X>7</X>
                        <Y>15</Y>
                </Point>
        </LinePoints>
        </ArcConnectorProperties>
        <ArcConnectorProperties>
            <Component>
                <ID>7</ID>
                <Name>S7</Name>
                <ConnectionPoint>
                    <X>7</X>
                    <Y>6</Y>
                </ConnectionPoint>
            </Component>
            <Component>
                <ID>3</ID>
                <Name>3</Name>
                <ConnectionPoint>
                    <X>5</X>
                    <Y>6</Y>
                </ConnectionPoint>
            </Component>
            <LinePoints>
            </LinePoints>
        </ArcConnectorProperties>
        <ArcConnectorProperties>
            <Component>
                <ID>9</ID>
                <Name>S9</Name>
                <ConnectionPoint>
                    <X>17</X>
                    <Y>13</Y>
                </ConnectionPoint>
            </Component>
            <Component>
                <ID>6</ID>
                <Name>Filter</Name>
                <ConnectionPoint>
                    <X>19</X>
                    <Y>13</Y>
                </ConnectionPoint>
            </Component>
            <LinePoints>
            </LinePoints>
        </ArcConnectorProperties>
    </ListOfArcConnectors>
    <ListOfArcConnectedValves>
        <ConnectedValves>
            <ID>1</ID>
            <Valve>
                <ValveID>1</ValveID>
                <Object>Mixer</Object>
                <ObjectID>4</ObjectID>
            </Valve>
            <LineSegments>
                <LineSegment>
                <Point>
                    <X>33</X>
                    <Y>0</Y>
                </Point>
                <Point>
                    <X>33</X>
                    <Y>9</Y>
                </Point>
                </LineSegment>
            </LineSegments>
        </ConnectedValves>
        <ConnectedValves>
            <ID>2</ID>
            <Valve>
                <ValveID>2</ValveID>
                <Object>Mixer</Object>
                <ObjectID>4</ObjectID>
            </Valve>
            <LineSegments>
                <LineSegment>
                <Point>
                    <X>36</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>36</X>
                <Y>9</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>3</ID>
        <Valve>
            <ValveID>3</ValveID>
            <Object>Mixer</Object>
            <ObjectID>4</ObjectID>
        </Valve>
        <LineSegments>
            <LineSegment>
            <Point>
                <X>39</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>39</X>
                <Y>9</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>4</ID>
        <Valve>
            <ValveID>4</ValveID>
            <Object>Mixer</Object>
            <ObjectID>4</ObjectID>
        </Valve>
        <LineSegments>
            <LineSegment>
            <Point>
                <X>42</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>42</X>
                <Y>9</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>5</ID>
        <Valve>
            <ValveID>5</ValveID>
            <Object>Mixer</Object>
            <ObjectID>4</ObjectID>
        </Valve>
        <LineSegments>
            <LineSegment>
            <Point>
                <X>30</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>30</X>
                <Y>9</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>6</ID>
        <Valve>
            <ValveID>6</ValveID>
            <Object>Mixer</Object>
```

```
                <ObjectID>4</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
                <Point>
                        <X>27</X>
                        <Y>0</Y>
                </Point>
                <Point>
                        <X>27</X>
                        <Y>9</Y>
                </Point>
                </LineSegment>
        </LineSegments>
</ConnectedValves>
<ConnectedValves>
        <ID>7</ID>
        <Valve>
                <ValveID>7</ValveID>
                <Object>Mixer</Object>
                <ObjectID>4</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
                <Point>
                        <X>45</X>
                        <Y>0</Y>
                </Point>
                <Point>
                        <X>45</X>
                        <Y>9</Y>
                </Point>
                </LineSegment>
        </LineSegments>
</ConnectedValves>
<ConnectedValves>
        <ID>8</ID>
        <Valve>
                <ValveID>8</ValveID>
                <Object>S7</Object>
                <ObjectID>7</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
                <Point>
                        <X>24</X>
                        <Y>0</Y>
                </Point>
                <Point>
                        <X>24</X>
                        <Y>18</Y>
                </Point>
                </LineSegment>
                <LineSegment>
                <Point>
                        <X>24</X>
                        <Y>18</Y>
                </Point>
                <Point>
                        <X>23</X>
                        <Y>18</Y>
                </Point>
                </LineSegment>
        </LineSegments>
</ConnectedValves>
<ConnectedValves>
        <ID>9</ID>
        <Valve>
                <ValveID>9</ValveID>
                <Object>S7</Object>
                <ObjectID>7</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
```

```
            <Point>
                <X>18</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>18</X>
                <Y>18</Y>
            </Point>
            </LineSegment>
            <LineSegment>
            <Point>
                <X>18</X>
                <Y>18</Y>
            </Point>
            <Point>
                <X>21</X>
                <Y>18</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>10</ID>
        <Valve>
            <ValveID>10</ValveID>
            <Object>S7</Object>
            <ObjectID>7</ObjectID>
        </Valve>
        <LineSegments>
            <LineSegment>
            <Point>
                <X>21</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>21</X>
                <Y>18</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>11</ID>
        <Valve>
            <ValveID>11</ValveID>
            <Object>S8</Object>
            <ObjectID>8</ObjectID>
        </Valve>
        <LineSegments>
            <LineSegment>
            <Point>
                <X>48</X>
                <Y>0</Y>
            </Point>
            <Point>
                <X>48</X>
                <Y>18</Y>
            </Point>
            </LineSegment>
            <LineSegment>
            <Point>
                <X>48</X>
                <Y>18</Y>
            </Point>
            <Point>
                <X>51</X>
                <Y>18</Y>
            </Point>
            </LineSegment>
        </LineSegments>
    </ConnectedValves>
    <ConnectedValves>
        <ID>12</ID>
        <Valve>
```

```
                <ValveID>12</ValveID>
                <Object>S8</Object>
                <ObjectID>8</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
                <Point>
                        <X>54</X>
                        <Y>0</Y>
                </Point>
                <Point>
                        <X>54</X>
                        <Y>18</Y>
                </Point>
                </LineSegment>
                <LineSegment>
                <Point>
                        <X>54</X>
                        <Y>18</Y>
                </Point>
                <Point>
                        <X>53</X>
                        <Y>18</Y>
                </Point>
                </LineSegment>
        </LineSegments>
</ConnectedValves>
<ConnectedValves>
        <ID>13</ID>
        <Valve>
                <ValveID>13</ValveID>
                <Object>S8</Object>
                <ObjectID>8</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
                <Point>
                        <X>51</X>
                        <Y>0</Y>
                </Point>
                <Point>
                        <X>51</X>
                        <Y>18</Y>
                </Point>
                </LineSegment>
        </LineSegments>
</ConnectedValves>
<ConnectedValves>
        <ID>14</ID>
        <Valve>
                <ValveID>14</ValveID>
                <Object>S9</Object>
                <ObjectID>9</ObjectID>
        </Valve>
        <LineSegments>
                <LineSegment>
                <Point>
                        <X>53</X>
                        <Y>62</Y>
                </Point>
                <Point>
                        <X>53</X>
                        <Y>41</Y>
                </Point>
                </LineSegment>
        </LineSegments>
</ConnectedValves>
<ConnectedValves>
        <ID>15</ID>
        <Valve>
                <ValveID>15</ValveID>
                <Object>S9</Object>
                <ObjectID>9</ObjectID>
        </Valve>
```

```
                <LineSegments>
                    <LineSegment>
                    <Point>
                        <X>50</X>
                        <Y>62</Y>
                    </Point>
                    <Point>
                        <X>51</X>
                        <Y>62</Y>
                    </Point>
                    </LineSegment>
                    <LineSegment>
                    <Point>
                        <X>51</X>
                        <Y>62</Y>
                    </Point>
                    <Point>
                        <X>51</X>
                        <Y>41</Y>
                    </Point>
                    </LineSegment>
                </LineSegments>
            </ConnectedValves>
            <ConnectedValves>
                <ID>16</ID>
                <Valve>
                    <ValveID>16</ValveID>
                    <Object>S9</Object>
                    <ObjectID>9</ObjectID>
                </Valve>
                <LineSegments>
                    <LineSegment>
                    <Point>
                        <X>56</X>
                        <Y>62</Y>
                    </Point>
                    <Point>
                        <X>54</X>
                        <Y>62</Y>
                    </Point>
                    </LineSegment>
                    <LineSegment>
                    <Point>
                        <X>54</X>
                        <Y>62</Y>
                    </Point>
                    <Point>
                        <X>54</X>
                        <Y>41</Y>
                    </Point>
                    </LineSegment>
                    <LineSegment>
                    <Point>
                        <X>54</X>
                        <Y>41</Y>
                    </Point>
                    <Point>
                        <X>53</X>
                        <Y>41</Y>
                    </Point>
                    </LineSegment>
                </LineSegments>
            </ConnectedValves>
        </ListOfArcConnectedValves>
</Architecture>
```