Flow-Layer Physical Design for Microchips Based on Monolithic Membrane Valves

Jeffrey McDaniel, Brian Crites, Philip Brisk, and William H. Grover
University of California, Riverside

Editor's notes:
This article introduces a software toolchain for physical design and layout for the flow layer of microfluidic LoCs based on integrated microvalve technology. A case study shows that it can automatically produce layouts for the Mars Organic Analyzer LoC to detect biomolecules in soil on Mars.
—Tsung-Yi Ho, National Tsing Hua University

Laboratories on a chip (LoCs) based on integrated microvalve technology have been developed for a variety of biochemical applications, including low-cost point-of-care testing [12] and detection of organic matter on Mars [10]. Through automation and miniaturization, LoCs offer the benefits of higher throughput, lower sample/reagent usage, and reduced likelihood of human error compared to traditional benchtop chemistry methods. These chips can be viewed as miniaturized plumbing networks that have been shrunk down to the micrometer scale and below. A typical microvalve-based chip comprises two layers: a flow layer, which transports fluid, and the control layer, which delivers externally supplied pneumatic pressure to open and close microvalves as needed.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.
Digital Object Identifier 10.1109/MDAT.2015.2459699
Date of publication: 08 October 2015; date of current version: 06 November 2015.

At present, microvalve-based LoCs are designed and physically laid out by hand. This creates a high barrier to entry for any scientist who requires a new device to perform an experiment. The biochemical research must be put on hold while the LoC is designed, laid out, and fabricated. This is particularly arduous for scientists who are not device experts, and lack naturally applicable training on synergistic topics such as semiconductor VLSI design and layout. The objective of our work is to automate the design process of these LoCs; this article reports a successful attempt to automate the physical design and layout of the flow layer in a two-layer chip.

Technology overview

The most widely recognized microvalve technologies are elastomeric valves based on multilayer soft lithography, developed at Stanford University [1], and monolithic membrane valves developed at the University of California Berkeley [4]. Although the software platform that we are developing is technology independent, we target the University of California Berkeley monolithic membrane valves, shown in Figure 1. LoCs based on monolithic membrane valves are built using two glass plates that sandwich a thin layer of polydimethylsiloxane

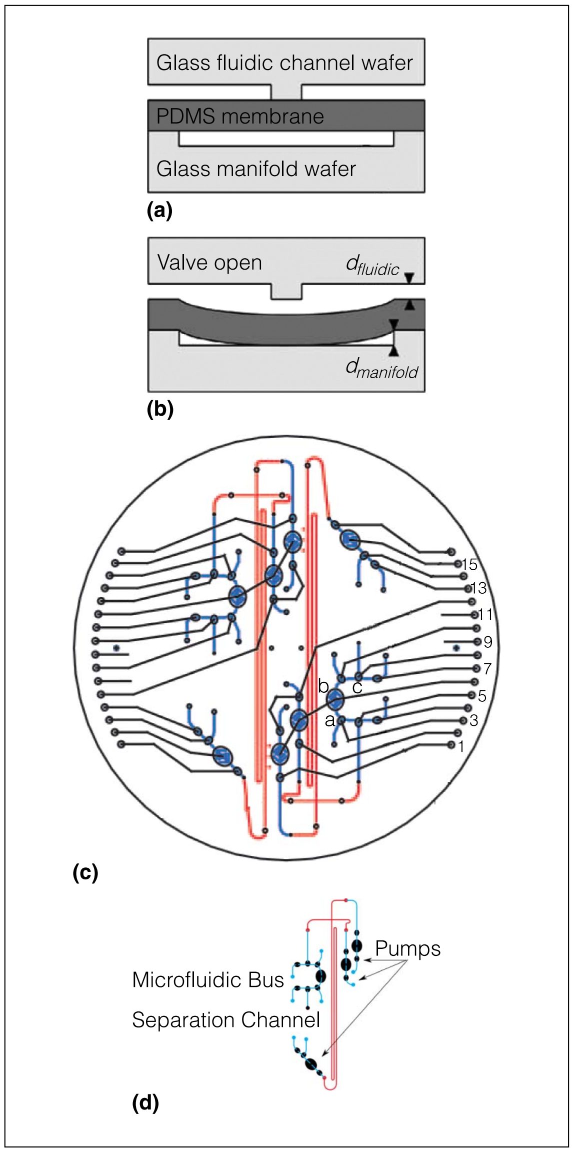


Figure 1. (a) The monolithic membrane valve developed by Grover et al. [4] is initially closed. When a vacuum is applied to the channel on the manifold side (b), the membrane is deflected, allowing fluid to flow through the valve. (c) Original chip designed for the Mars rover and (d) a single component extracted from the chip [10].

(PDMS), a flexible and inert organic polymer. Etched channels in the two glass plates, respectively, provide distinct layers for fluid and pneumatic control. The biochemical reaction executes on the flow layer, while the control layer delivers pressure to each microvalve to control fluids in the flow layer.

Monolithic membrane vales are normally closed, as illustrated in Figure 1a. Applying a vacuum through a control channel deflects the membrane, which opens the valve, and allows fluid to flow through, as shown in Figure 1b. The Mars organic analyzer (MOA) [10], shown in Figure 1c, is a representative LoC built using monolithic membrane valves. Both the fluid flow and control layers of the MOA were designed and physically laid out by hand. The MOA contains two copies of the same basic analysis system for the purpose of fault tolerance and redundancy. Figure 1d depicts the fluid flow layer (control lines removed) of one of the two analysis systems.

Background

In our framework, a biological experiment is specified using a domain-specific language suitable for the chosen technology; architectural synthesis, which includes scheduling, resource allocation, and binding steps, converts this specification into a graph-based netlist (plumbing network) capable of executing the experiment [7], [9]. If desired, the netlist can be converted to the microfluidic hardware description language (MHDL), which is human readable representation. MHDL is extensible, allowing the user to describe both the technology and architectural entities within their own respective library files [6].

The focus of this article is the next step, which is to automatically convert the MHDL or netlist representation of the LoC architecture into a physical layout of the fluid flow layer. Having one layer for fluid flow imposes the constraint that only planar LoC architectures can be realized in this technology. It is possible to planarize a nonplanar architecture by inserting microvalves to act as switches at fluid channel intersection points [8]; however, doing so is problematic because additional external control lines are required to actuate the switches. The number of external control lines is typically limited as a design rule, and adding more control lines tends to reduce reliability after fabrication. To keep matters simple here, we limit the discussion here to physical design for planar LoC architectures; the possibility of automatic planarization is left open for future work.

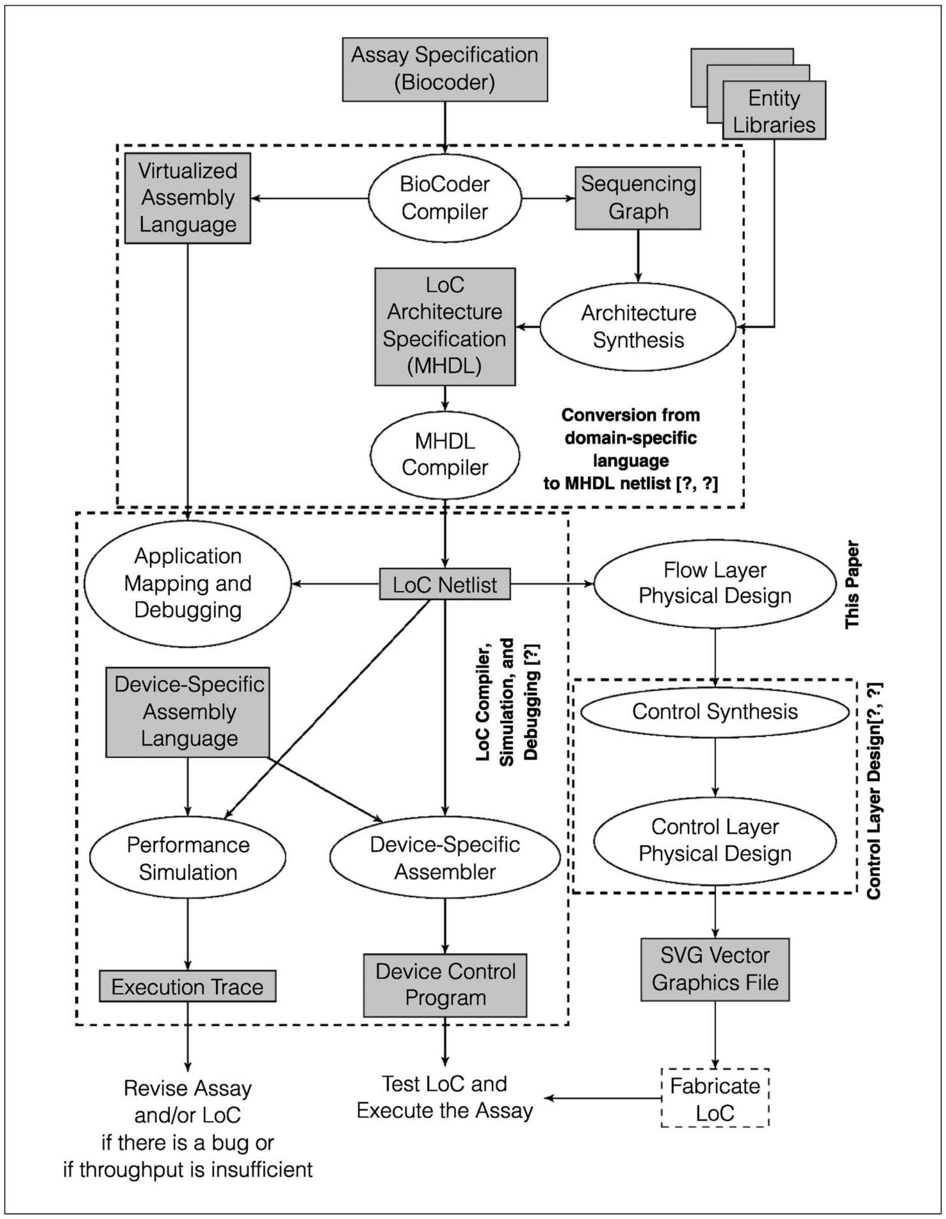
Prior work on flow layer component placement uses simulated annealing [8], [9], which is based on randomization and iterative improvement. Simulated annealing does not guarantee a planar layout, even if the netlist being placed is planar. In our opinion, simulated annealing could be used as a postprocessing step to reduce the area or fluid channel length of a precomputed planar layout, while ensuring that the optimized layout remains planar as a constraint. Our system, in contrast, uses a greedy approach, which eschews randomization, for postprocessing, while ensuring by construction that the resulting layout remains planar.

The planar layout for the fluid flow layer is converted to a scalable vector graphic (SVG) file, which can be used to create a mask that produces patterns for etched channels in one of the two glass layers. After laying out the fluid layer, we manually design, lay out, and produce an SVG file for the pneumatic control layer as well; future work will integrate existing algorithms that effectively automate control layer generation [5], [9] into our toolflow. To fabricate a fully integrated device, separate SVG files are required for the fluid flow and pneumatic control layers. To date, no working toolflow that can produce both flow and/or control layer designs has been reported in the literature.

System overview

Figure 2 illustrates the main stages of our software toolflow. To make the system flexible and extensible, technology library files are used to specify the available LoC technologies and their corresponding entities. Extensibility allows device engineers to continuously extend our toolflow whenever they develop new technologies and/or entities. The technology library file describes fabrication constraints, while the entity library files specify the capabilities and constraints of each component [6]. During component expansion, in which vertices (points) in the netlist are replaced with 2-D microfluidic components, the placer obtains the dimensions of each component from its entity library file. The router ensures the routing channels are aligned with each entity's ports and ensures proper spacing to allow for legal fabrication.

The input to our physical design algorithm is a planar netlist of components and their fluidic connections, a description of the target technology, and the entities to be used. The netlist can be generated from an MHDL specification [6] or from a synthesis



Assay Specification (Biocoder)
Entity Libraries
Virtualized Assembly Language
BioCoder Compiler
Sequencing Graph
LoC Architecture Specification (MHDL)
Architecture Synthesis
MHDL Compiler
Conversion from domain-specific language to MHDL netlist [?, ?]
Application Mapping and Debugging
LoC Netlist
Flow Layer Physical Design
This Paper
Device-Specific Assembly Language
LoC Compiler, Simulation, and Debugging [?]
Control Synthesis
Control Layer Physical Design
Control Layer Design[?, ?]
Performance Simulation
Device-Specific Assembler
Execution Trace
Device Control Program
SVG Vector Graphics File
Revise Assay and/or LoC if there is a bug or if throughput is insufficient
Test LoC and Execute the Assay
Fabricate LoC

Figure 2. The main stages of our software toolflow. This article focuses on the flow-layer physical-design stage. Our future work will be integrating the control synthesis and physical design from [5] and [7].

tool starting from a high-level domain-specific language [7], as shown in Figure 2.

The netlist is initially treated as a graph in which vertices are points, as opposed to physical components that have 2-D areas. The netlist is placed using a straight-line planar embedding algorithm [3]. Nodes are then expanded from points to 2-D components, based on their entity types. Flow channels between components are routed using a modified variant of an established semiconductor very large scale integration (VLSI) router [11]. Last, a postprocessing step adjusts the placement solution and incrementally reroutes the chip in order to reduce area and fluid routing channel length.

Flow-layer placement

Planar embedding

Component placement starts by computing a straight-line planar embedding. The netlist is represented as a graph $G = (V, E)$, where V is a set of components (without dimensions and/or area) and E is a set of fluid channels connecting components. First, the Boyer–Myrvold method [2] is applied to make G fully connected and to test for planarity. If G is planar, then it is transformed to be biconnected and maximally planar. The vertices $v_i \in V$ are then ordered canonically, which enables linear-time computation of a straight-line planar embedding of the netlist on a $(2|V|-4) \times (|V|-2)$ grid [3]. Algorithm 1 presents the pseudocode.

Algorithm 1: Chrobak–Payne straight-line embedding from the Boost library. The function calls shown here are Boost library calls.

```
Require: $G := (V, E)$ an undirected graph
Ensure: $G := (V, E)$ with each $v_i \in V$ placed
1: $G := make\_connected(G)$
2: if $!boyer\_myrvold\_planarity\_test(G)$ then
3:     $exit()$
4: endif
5: $G := make\_biconnected\_planar(G)$
6: $G := make\_maximal\_planar(G)$
7: $X := planar\_canonical\_ordering(G)$
8: $G := chrobak\_payne\_straight\_line(G, X)$
```

Component expansion

The straight-line planar embedding does not account for size or dimensions of components. To create a valid placement, we apply two passes to expand components and remove any overlap between them that may result. The first pass sorts the components $c_i \in C$ by their x-coordinates, x_i, in ascending order, and expands each component by its width w_i. All subsequent components $c_j \in C$, where $j > i$, are shifted in the positive x-direction by w_i; the new position is $x'_j = x_j + w_i$. The second pass of the expansion applies the same steps along the y-axis, while expanding and shifting components based on their heights, rather than their widths.

Component expansion cannot preserve the straight-line planar embedding property, illustrated in Figure 3. Additionally, there is no direct mechanism to assign fluid channels to ports on the perimeter of each component. Our flow layer router, described next, addresses these challenges.

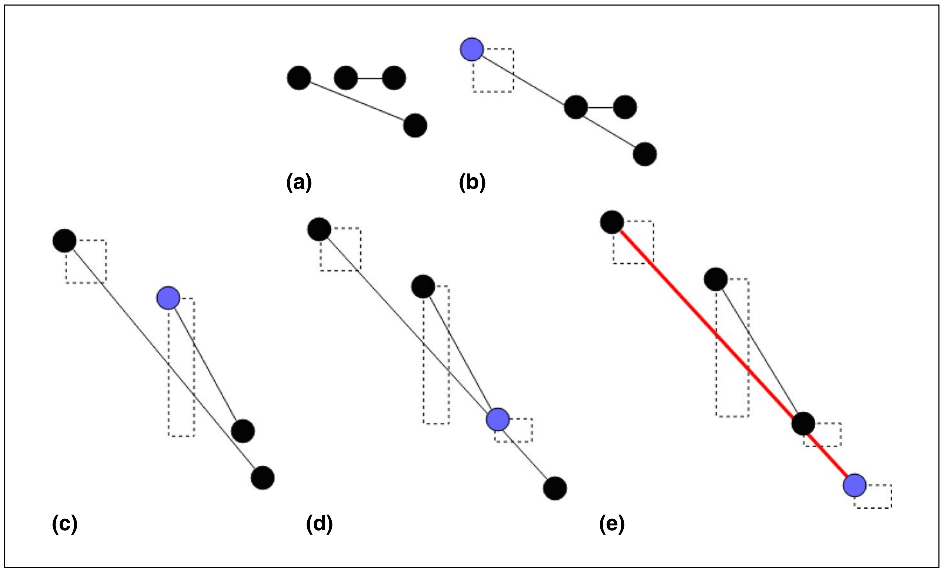


Figure 3. (a) The original graph; (b)–(e) expand the components one at a time to their full size. The bold red line represents the straight-line connection that has been invalidated due to the expansion.

Flow-layer routing

The next step is to instantiate a routing grid $R = (U, F)$, where

U is a set of grid points, and F is a set of edges representing potential channel routes between adjacent grid points. For each component $c_i \in C$, a vertex u_i for the ports $p_h \in c_i$ is instantiated and added to U. A grid of vertices is then instantiated in the empty space between components. The pseudocode is presented in Algorithm 2. In lines 16 and 17, edges that represent potential routing channel segments are added to F by instantiating a bidirectional edge f_i with a capacity of 1 between distinct vertices $u_i \in U$ and $u_j \in U$ if and only if $(u_j.x - u_i.x == 1) \oplus (u_j.y - u_i.y == 1)$.

Algorithm 2: Grid creation.

```
Require: $C$ := set of components in the netlist
Require: $\mathrm{Max}_x, \mathrm{Max}_y$ are the maximum $x$ and $y$ values in the plane
Ensure: $R := (U, F)$ grid of vertices
 1: for all $c_i \in C$ do
 2:   for all $p_h \in c_i$ do
 3:     $U \leftarrow U \cup \{u_i = (p_h.x, p_h.y)\}$
 4:   endfor
 5: endfor
 6: for all $0 < x < \mathrm{Max}_x$ do
 7:   for all $0 < y < \mathrm{Max}_y$ do
 8:     if $!within\_component(x, y)$ then
 9:       $U \leftarrow U \cup \{u_i = (x, y)\}$
10:     endif
11:   endfor
12: endfor
13: for all $0 < x < \mathrm{Max}_x$ do
14:   for all $0 < y < \mathrm{Max}_y$ do
15:     $u_i \leftarrow (x, y)$
16:     $F \leftarrow F \cup get\_south\_neighbor(u_i)$
17:     $F \leftarrow F \cup get\_east\_neighbor(u_i)$
18:   endfor
19: endfor
```

The network flow model, described next, ensures that no edge is used more than once. To ensure that no vertex is used more than once, each vertex $u_i \in U$ is split into u_i' and u_i'' and a directed edge $f_i = (u_i, u_i'')$ is added to F. All incoming edges to u_i are replaced with edges into u_i' and all outgoing edges from u_i are replaced with edges leaving u_i''. Hence, any fluid channel that routes through u_i must now use edge f_i. The edge capacity constraint ensures that at most one such channel may use the vertex.

Network flow model

The next step is to route channels between the components using a network flow routing method based on [11]. Components are processed in order, and unrouted channels that are incident on each component are routed together. Special nodes (super sources, supersinks, and sink groups) are added to the routing problem to enable the network flow to simultaneously route and perform port assignment. We set up our network flow algorithm from the source component c_i to its set of sink components T_i as described in Algorithm 3.

Algorithm 3: Network flow model for channel routing.

```
Require: $C$ := set of components in the netlist
Ensure: $R := (U, F)$ is a network flow model
 1: $U \leftarrow \{u_{\mathrm{supersource}}, u_{\mathrm{supersink}}\}$
 2: $F \leftarrow \phi$
 3: for all $t_j \in T_i$ do
 4:   $U \leftarrow U \cup \{= u_{\mathrm{sink\_group}_{t_j}}\}$
 5:   $F \leftarrow F \cup \{f_j = (u_{\mathrm{sink\_group}_{t_j}}, u_{\mathrm{supersink}})\}$
 6:   $capacity(f_j) \leftarrow 1$
 7:   $cost(f_j) \leftarrow 1$
 8:   for each port $p_k \in P_i$ of $t_j$ do
 9:     $U \leftarrow U \cup \{u_{p_k}\}$
10:     $F \leftarrow F \cup \{f_{p_k} = (u_{\mathrm{sink\_group}_{t_j}}, u_{p_k})\}$
11:     $capacity(f_{p_k}) \leftarrow 1$
12:     $cost(f_{p_k}) \leftarrow 0$
13:   endfor
14: endfor
15: for each port $p_j \in P_i$ of $c_i$ do
16:   $U \leftarrow U \cup \{u_{p_j}\}$
17:   $F \leftarrow F \cup \{f_{p_j} = (u_{\mathrm{supersource}}, u_{p_j})\}$
18:   $capacity(f_{p_j}) \leftarrow 1$
19:   $cost(f_{p_j}) \leftarrow 0$
20:   endfor
```

A set of routes from c_i to all $t_j \in T_i$ is found by computing the maximum flow from $u_{\mathrm{supersource}}$ to $u_{\mathrm{supersink}}$. The paths computed by the network flow algorithm include port assignment at the source and sinks, and may present multiple valid paths. As shown in Figure 4, we then trace the path from the port p_k at each sink t_i to its corresponding port p_j at the source component c_i, as determined by the solution to the network flow problem. This traceback obtains the shortest valid path found. The supersource, supersink,

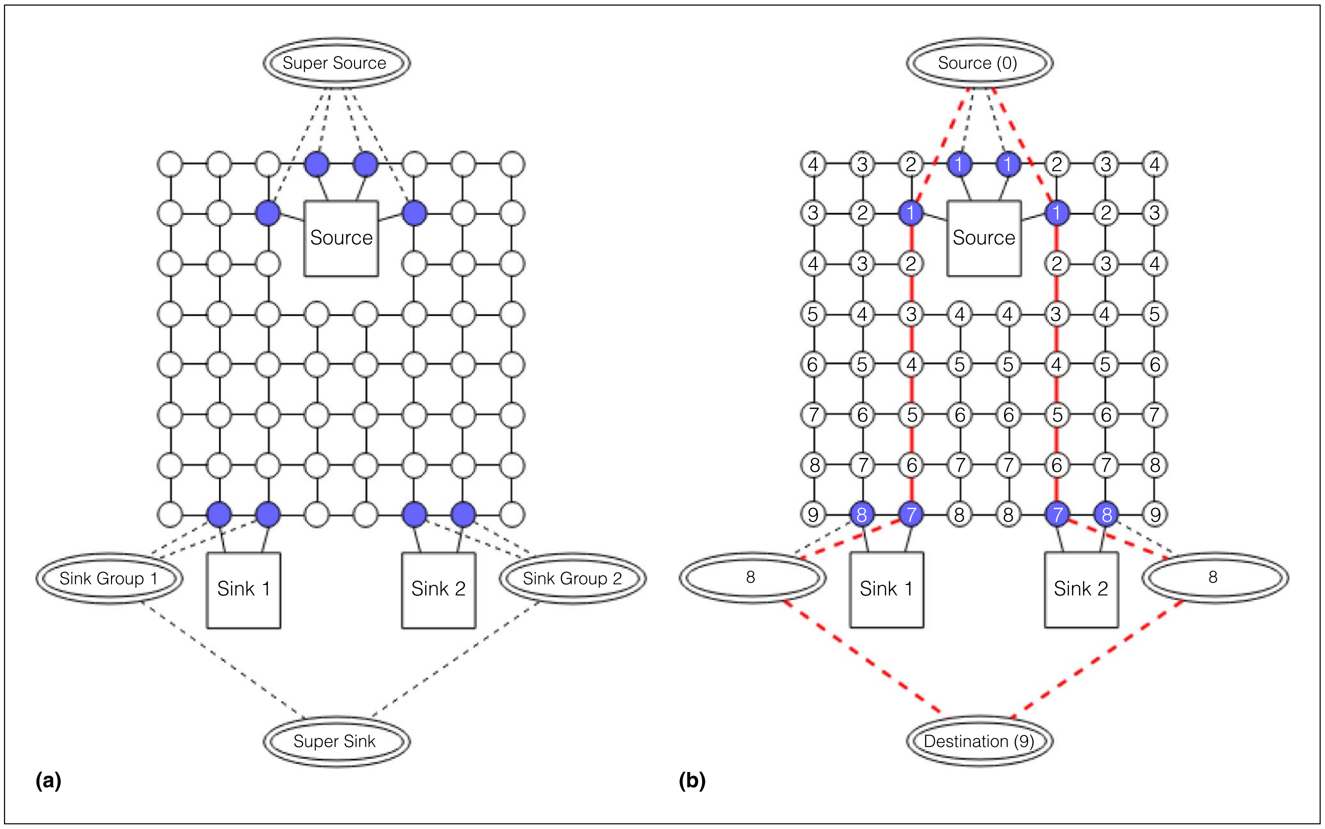


Figure 4. (a) The supersource, sink group, and supersink nodes are added to the grid. (b) A minimum-cost maximum-flow network algorithm is run on the system to find all necessary edges from source to sink components; the example resulting paths are shown in red.

and sink groups along with their incident edges are then removed from the routing grid, and the process repeats for the next component.

Our approach offers two enhancements to the existing network flow router [11] which improve routability. First, if a route between components c_i and c_j abuts a third component c_k, then the ports on c_k may become blocked. To prevent blockage, we create a buffer zone of a few vertices around each component. Vertices within the buffer zone are removed from the routing grid to prevent port blockage; they are returned to the grid only when routing that component. This ensures that each connection will be able to at least find a port to route out of the component.

Second, routing failures may occur due to fracturing of the routing grid as more connections are routed. If a routing failure occurs, all routes are removed and the queue of components is reordered so that the component that failed to route now routes first; this guarantees that the component will now be able to route. We limit the number of times that the component queue may be reordered; if this limit is exceeded, we declare a routing failure for the chip. No routing failures were observed in our experiments.

Postprocessing

The straight-line planar embedding algorithm that we use [3] is not cognizant of component dimensions and makes no attempt to reduce the area in terms of grid usage. Manual inspection of physically laid out chips demonstrates ample opportunities to reduce area and fluid channel length.

As a postprocessing step, we search for opportunities to move components placed on the chip's perimeter into the interior without sacrificing the planar layout property. One by one, we select modules placed on the perimeter of the chip and attempt to move them in a direction orthogonal to the perimeter with which they are aligned. After a component is moved, its incident channels are rerouted. If a legal route is obtained, then the movement is accepted; otherwise, it is rejected. The process repeats until no further movements that reduce chip can be found. A detailed example is shown in the following section.

The physical space on the chip is discretized using a grid graph, as discussed previously. Let N be the number of hops (graph nodes) that a component can be moved in one direction without compromising the (nonrouting) legality of the resulting placement. If the resulting route is legal, then we are done; otherwise, we try again. To ensure rapid convergence, we employ a binary search. If a legal placement and routing solution is not found at N hops, then we try again at $N/2$ hops, etc. This ensures that a legal placement and routing solution is found for each perimeter component after $O(\log N)$ routing attempts. This process repeats until no perimeter components can be moved into the interior of the chip.

Results

We created entity library files using components from the original MOA chip [10]. We extracted a netlist for the MOA and specified it in MHDL. We then ran the chip through our placement and router, yielding a workable flow layer; we manually routed the pneumatic control layer. Figure 5a shows the resulting device layout. The bounding boxes represent component dimensions used during placement and routing, and were replaced by the actual component image in the SVG file that was generated. It is clear from a straightforward visual inspection that numerous local perturbations to the component layout could reduce chip area and/or fluid routing channel length. Figure 5b shows the resulting device layout after applying our postprocessing step, which yields a more compact planar layout; we manually rerouted the control layer shown in the figure. These results effectively demonstrate the ability of our toolflow to adapt physical-design algorithms originally developed for semiconductor VLSI, to microvalve-based LoCs.

This article has presented the first automated toolchain that can convert a netlist representation of the flow-layer microvalve-based LoC into a physical

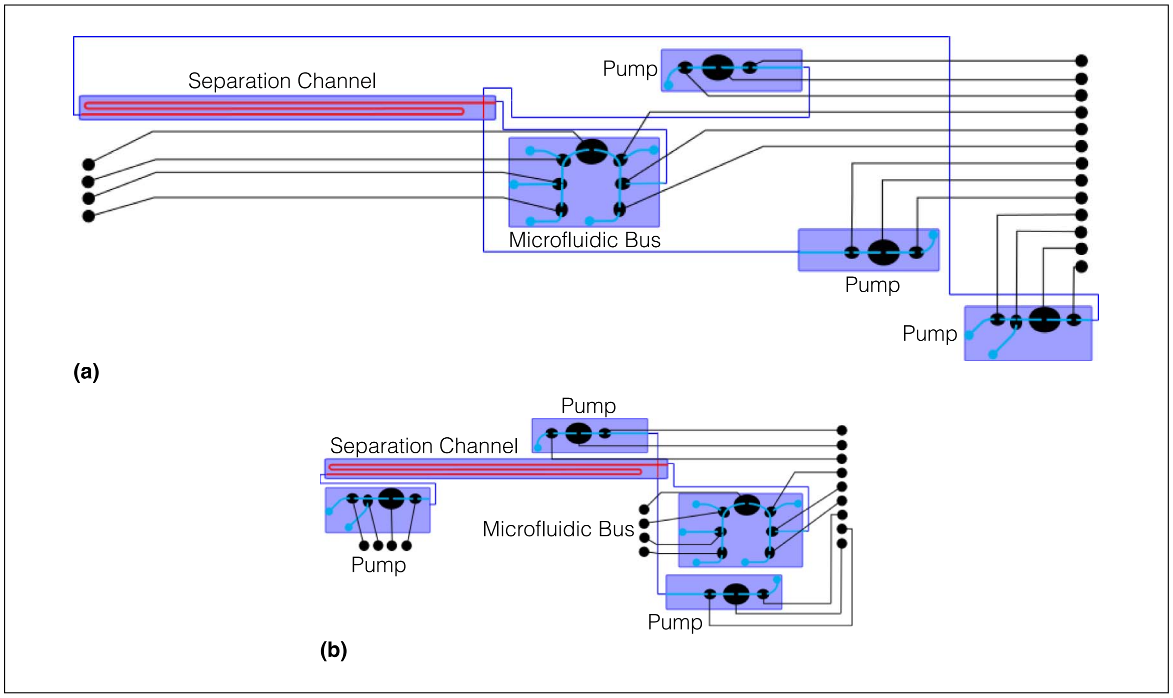


Figure 5. (a) The assembled MOA chip [10] placed and routed using our algorithm. The control layer has been manually routed, after the flow layer SVG was automatically generated. The bounding boxes are used during our algorithms; we show them here filled in with original components as described by the entity library files. (b) After a postprocessing step, and rerouting the control layer, we are able to reduce the overall size of the final chip.

layout that can be fabricated. Future work will provide comparable automation for the pneumatic control layer. We also plan to investigate more effective straight-line planar embedding algorithms that optimize for more efficient grid usage and can handle 2-D components. Automatic planarization of nonplanar netlists is another important area for future work. ■

References

[1] I. E. Araci and S. R. Quake, "Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves," *Lab Chip*, vol. 12, no. 16, pp. 2803–2806, Aug. 2012.

[2] J. M. Boyer and W. J. Myrvold, "On the cutting edge: Simplified O(n) planarity by edge addition," *J. Graph Algorithms Appl.*, vol. 8, no. 3, pp. 241–273, 2004.

[3] M. Chrobak and T. H. Payne, "A linear-time algorithm for drawing a planar graph on a grid," *Inf. Process. Lett.*, vol. 54, pp. 241–246, 1995.

[4] W. H. Grover, A. M. Skelley, C. N. Liu, E. T. Lagally, and R. A. Mathies, "Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices," *Sens. Actuators B, Chem.*, vol. 89, no. 3, pp. 315–323, Apr. 2003.

[5] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer optimization for flow-based mVLSI microuidic biochips," in *Proc. Int. Conf. Compilers Architect. Synthesis Embedded Syst.*, Oct. 12–17, 2014, DOI: 10.1145/2656106.2656118.

[6] J. McDaniel, A. Baez, B. Crites, A. Tammewar, and P. Brisk, "Design and verification tools for continuous fluid flow-based microfluidic devices," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 22–25, 2013, pp. 219–224.

[7] J. McDaniel, C. Curtis, and P. Brisk, "Automatic synthesis of microfluidic large scale integration chips from a domain-specific language," in *Proc. Biol. Circuits Syst.*, Oct. 31–Nov. 2, 2013, pp. 3–6.

[8] J. McDaniel, B. Parker, and P. Brisk, "Simulated annealing-based placement for microfluidic large scale integration (MLSI) chips," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr.*, Oct. 6–8, 2014, pp. 213–218.

[9] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga, "Architectural synthesis of flow-based microfluidic large-scale integration biochips," in *Proc. Int. Conf. Compilers Architect. Synthesis Embedded Syst.*, Oct. 7–12, 2012, pp. 181–190.

[10] A. M. Skelley et al., "Development and evaluation of a microdevice for amino acid biomarker detection and analysis on Mars," *Proc. Nat. Acad. Sci. USA*, vol. 102, no. 4, pp. 1041–1046, 2005.

[11] M. D. Wong, H. Xiang, and X. Tang, "Min-cost flow-based algorithm for simultaneous pin assignment and routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 7, pp. 870–878, Jul. 2003.

[12] P. Yager et al., "Microfluidic diagnostic technologies for global public health," *Nature*, vol. 442, no. 7101, pp. 412–418, Jul. 2006.

Jeffrey McDaniel is currently working toward a PhD in computer science at the University of California at Riverside (UCR), Riverside, CA, USA. To date, he has published five conference papers and two journal papers. His research interests include languages, synthesis, and hardware interfacing for continuous-flow microfluidic biochips. McDaniel has a BS in computer science and pure mathematics and an MS in computer science from UCR (2011 and 2014, respectively).

Brian Crites is currently working toward a PhD in computer science at the University of California at Riverside (UCR), Riverside, CA, USA. To date, he has published two conference papers. His research interests include graph theory and synthesis and hardware interfacing for continuous-flow microfluidic biochips. Crites has a BS in computer engineering and an MS in computer science from UCR (in 2012 and 2014, respectively).

Philip Brisk has been an Associate Professor at the Department of Computer Science and Engineering, Bourns College of Engineering, University of California Riverside, Riverside, CA, USA, since July 1, 2015. From 2006 to 2009, he was a Postdoctoral Scholar in the Processor Architecture laboratory, School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, (EPFL), Lausanne, Switzerland. From 2009 to 2015, he was an Assistant Professor in the Department of Computer Science and Engineering, Bourns College of Engineering, University of California Riverside. His research interests include field-programmable gate arrays (FPGAs), compilers, and design automation and architecture for application-specific processors. Brisk has a BS, an MS, and a PhD in computer science

from the University of California Los Angeles (UCLA), Los Angeles, CA, USA (2002, 2003, and 2006, respectively). He was a recipient of the Best Paper Award at the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES 2007) and the 2009 International Conference on Field-Programmable Logic and Applications (FPL 2009). He is or has been a member of the program committees of several international conferences and workshops, including Design Automation Conference (DAC), Asia and South Pacific Design Automation Conference (ASPDAC), Design, Automation & Test in Europe Conference & Exhibition (DATE), International Conference on Very Large Scale Integration (VLSI-SoC), FPL, International Conference on Field-Programmable Technology (FPT), and others. He was the General (Co-)Chair of the 2009 IEEE Symposium on Industrial Embedded Systems (SIES 2009), the 2010 IEEE Symposium on Application Specific Processors (SASP 2010), and the 2011 International Workshop on Logic and Synthesis (IWLS 2011), and participated in the organizing committee of many other conferences and symposia as well.

William H. Grover is an Assistant Professor in the Department of Bioengineering, Bourns College of Engineering, University of California at Riverside, Riverside, CA, USA. At UCR, his lab develops microfluidic instruments for analyzing the fundamental physical properties (mass, volume, density, etc.) of single living cells and microorganisms. His lab also explores the interface between microfluidics and computer science to develop "programmable microfluidics" that can perform many different functions by simply running different "programs." Prior to joining UCR, he received his postdoctoral training in the Biological Engineering Division at the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. At MIT, he developed a microfluidic technique for making the first precision measurements of the density of single living cells. At University of California Berkeley, Berkeley, CA, USA, he developed the first microfluidic valves suitable for large-scale use in glass microfluidic devices. Now cited over 400 times and the subject of several issued U.S. patents, this valving technology is the technological foundation of one company (IntegenX, Pleasanton, CA, USA) and several academic research labs. Grover has a PhD in chemistry from the University of California Berkeley.

■ Direct questions and comments about this article to Jeffrey McDaniel and Brian Crites, Department of Computer Science and Engineering, University of California at Riverside, Riverside, CA 92507 USA; jmcda001@ucr.edu.