Computer-Aided Design for Microfluidic Chips Based on Multilayer Soft Lithography

Nada Amin[1], William Thies[2] and Saman Amarasinghe[1]
[1] Massachusetts Institute of Technology
[2] Microsoft Research India

***Abstract*— Microfluidic chips are emerging as a powerful platform for automating biology experiments. As it becomes possible to integrate tens of thousands of components on a single chip, researchers will require design automation tools to push the scale and complexity of their designs to match the capabilities of the substrate. However, to date such tools have focused only on droplet-based devices, leaving out the popular class of chips that are based on multilayer soft lithography.**

In this paper, we develop design automation techniques for microfluidic chips based on multilayer soft lithography. We focus our attention on the control layer, which is driven by pressure actuators to invoke the desired flows on chip. We present a language in which designers can specify the Instruction Set Architecture (ISA) of a microfluidic device. Given an ISA, we automatically infer the locations of valves needed to implement the ISA. We also present novel algorithms for minimizing the number of control lines needed to drive the valves, as well as for routing valves to control ports while admitting sharing between the control lines.

To the microfluidic community, we offer a free computer-aided design tool, Micado, which implements a subset of our algorithms as a practical plug-in to AutoCAD. Micado is being used successfully by microfluidic designers. We demonstrate its performance on three realistic chips.

I. Introduction

Microfluidic chips are "lab-on-a-chip" systems that can automate biology experiments by programmatically manipulating small quantities of fluids [1], [2]. Microfluidics is a diverse field, with many competing technologies for implementing the chips themselves. In the design automation community, most of the attention thus far has been placed on droplet-based processors that manipulate fluids on an electrode array [3], [4], [5]. However, a competing technology that is popular amongst many scientists is that of multilayer soft lithography, where fluids flow along predefined channels and are controlled by pressurized pumps and valves [6], [7]. The technology for manufacturing microfluidic chips using soft lithography has advanced faster than Moore's Law [8]; today, there is a commercially-available chip that uses over 25,000 valves and about a million features to run 9,216 polymerase chain reactions in parallel [9], [10].

Despite these advances, the design methodology for microfluidic chips relies on many manual steps and represents a serious barrier to scaling the design complexity to the limits allowed by the underlying technology. Researchers typically design chips by drawing them in AutoCAD, with placement and routing done by hand. The control logic is manually orchestrated to accomplish the steps needed to perform an experiment, and graphical user interfaces (GUIs) are re-constructed in a separate program (LabView) that is disconnected from the chip layout. This manual design process does not scale and is very brittle to design changes; for example, adding a few valves often entails complete re-routing of the chip and re-design of the GUI.

Our vision is to bring the same automation and discipline to the microfluidic design process that electronic CAD brought to circuit design. While researchers have developed techniques to automate the mapping of biology experiments to droplet-based fluidic processors [11], [12], [13], we are unaware of any research on design automation for microfluidic chips based on multilayer soft lithography.

As a first step towards this vision, in this paper we address the problem of generating the *control layer* on a microfluidic chip. As depicted in Figure 1, a chip manufactured with multilayer soft lithography consists of two layers: a flow layer and a control layer. Channels on the flow layer carry the biological fluids of interest, while channels on the control layer are connected to external pressure actuators. Designing the control layer requires three steps: 1) placing *valves* on top of the flow layer, which restrict fluid flow upon being pressurized; valve placement depends on the flow patterns that are required by the biology experiment, 2) placing *control ports* on the periphery of the chip, where external pressure actuators are inserted, and 3) routing each valve to a control port, via a control channel. The control layer is one of the most tedious aspects for designers today, as it requires careful reasoning and also needs to be repeated for every change to the flow topology or logical chip operation. The control layer is also a good target for automation, as it is subject to a well-defined set of design rules.

We describe a tool called Micado that automates the generation of the control layer on multilayer microfluidic

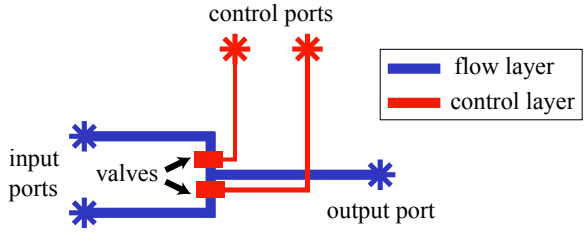


Fig. 1. A simple microfluidic chip based on multilayer soft lithography.

1. Control Inference. Given a drawing of the flow layer and an annotation of which flow paths are desired, Micado automatically places valves on the control layer to direct the given flows. Valves are shared when possible to decrease the number of control lines. (Note: while flows are annotated textually in the figure, they are indicated graphically in Micado.)

2. Routing. After the designer has indicated the positions of external control ports on the chip, Micado automatically connects each control line to a control port while respecting the design rules. The total length of the routes is minimized. Also, the number of corners (changes of direction) of control lines is reduced to yield an aesthetic outcome for the designer.

3. GUI Generation. Micado exports a graphical user interface for operating a chip in the laboratory. In AutoCAD, the user draws buttons (shown in green) and graphically associates them with flows of interest. When a button is clicked at runtime, valves are toggled to enable the associated flow. At right, the clicked button routes fluids from the last input, through the bottom of the mixer, to the output. Open valves are shown in blue, while closed valves are shown in red.

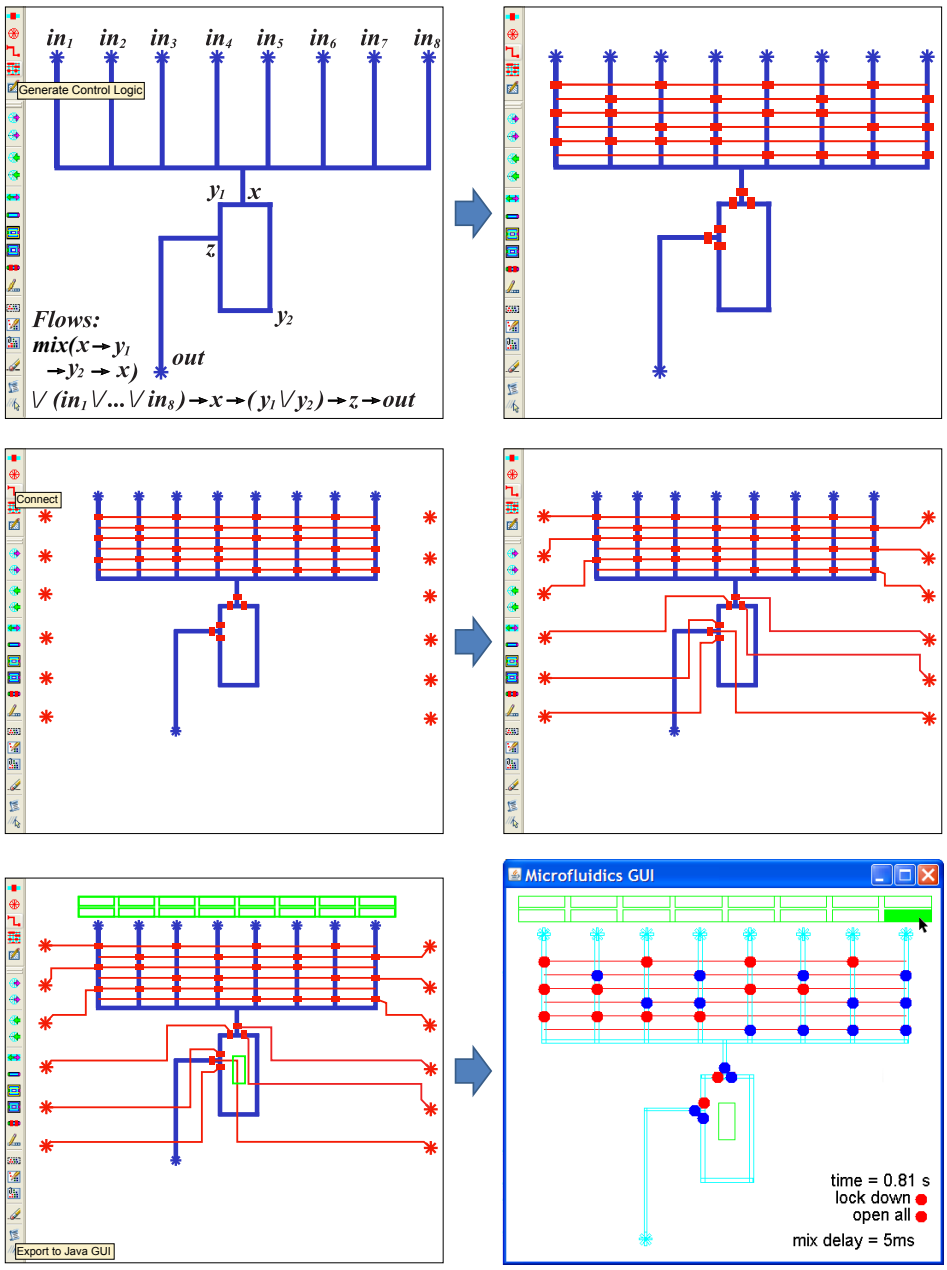


Fig. 2. Micado automates three key steps in the microfluidic design process: control inference, routing, and GUI generation.

chips. Implemented as an AutoCAD plugin and freely available online [14], the operation of Micado is illustrated by example in Figure 2. First, Micado allows the designer to specify the logical operation of a chip in the form of an instruction set architecture (ISA). The ISA specifies which combinations of flow channels should be active at a given time. Based on the ISA, Micado infers the placement of valves on the control layer. It also performs a novel analysis that determines which valves can share the same control line, which is important for reducing the number of external control ports (a bottleneck to the scalability of microfluidic chips). Next, Micado automatically routes each valve to a control port, while allowing certain valves to share control lines as derived previously. Finally, Micado generates a GUI for the chip, enabling the designer to interactively activate each desired flow pattern at runtime.

To summarize, this paper makes the following contributions:

- A language for specifying a fluidic ISA, the desired patterns of flow activation on a microfluidic chip (Section II).
- The first algorithm for inferring valve placement and control logic needed to implement a fluidic ISA. We prove that the general problem is NP-hard, giving rise to the need for heuristics (Section III).
- The first polynomial-time routing algorithm that connects each internal feature to one of many external ports

Flow Language			Input Point	Output Point	Constraints
ISA	:=	F	in(F)	out(F)	in(F) = *source* and out(F) = *sink*
F	:=	$P_1 \rightarrow P_2$	P_1	P_2	$P_1 \neq P_2$
	|	$F_1 \rightarrow F_2$	in(F_1)	out(F_2)	out(F_1) = in(F_2)
	|	$F_1 \vee F_2$	in(F_1)	out(F_1)	in(F_1) = in(F_2) and out(F_1) = out(F_2)
	|	$F_1 \wedge F_2$	in(F_1)	out(F_1)	in(F_1) = in(F_2) and out(F_1) = out(F_2)
	|	$F_1 \vee \text{mix}(F_2)$	in(F_1)	out(F_1)	in(F_2) = out(F_2)
	|	$F_1 \wedge \text{mix}(F_2)$	in(F_1)	out(F_1)	in(F_2) = out(F_2)
	|	pump(F)	in(F)	out(F)	

Fig. 3. Language for specifying a microfluidic ISA. For each method of composing flows, the table indicates the input and output points of the combined flow, as well as any constraints on the original flows.

while allowing configurable sharing between channels (Section IV).
- An implementation of our algorithms (some of them from an older version of this document [15]) as an AutoCAD plugin, Micado (Section V).
- An evaluation of Micado on real microfluidic chips, demonstrating that it is effective and useful in practice (Section VI).

In the remainder of this paper, we describe our techniques according to the outline above. We close by presenting related work (Section VII) and our conclusions (Section VIII).

II. Specifying a Microfluidic ISA

Previous research has established the notion of a microfluidic instruction set architecture (ISA) as the primitive set of operations supported by a microfluidic device [16], [17]. While ISA's for electronic chips are difficult to specify graphically (as the 32-bit logical operations are difficult to visualize), with microfluidics the instructions represent logical flows that can be hierarchically constructed from the graphical layout of the chip. We provide the first method to define an ISA as a hierarchical composition of flows, providing a simple methodology for specifying and analyzing a chip's functionality.

Our language for specifying the functionality of a chip appears in Figure 3. Our system can express three basic functions: the ability to flow fluids (in certain predefined patterns) from one location to another, the ability to pump fluids along a given path, and the ability to mix fluids (by pumping in a given circular path). Though we present the language as a context free grammar, in our implementation all primitives are indicated graphically in a CAD environment.

To specify a flow, the user begins by identifying points of interest on the flow layer of the microfluidic chip. Every flow F has a single start point $in(F)$ and a single end point $out(F)$, though the flow may split into many branches between the start and end points. Flows can be constructed using four primitives:

1) A simple flow $P_1 \rightarrow P_2$ connects points P_1 and P_2 directly, using the shortest path between them.
2) A sequential flow, $F_1 \rightarrow F_2$, connects two flows in sequence. The endpoint of F_1 must be the same as the starting point of F_2.
3) An OR-parallel flow, $F_1 \vee F_2$, indicates that fluids should pass through either F_1 or F_2 (but not both) depending on the runtime configuration. The flows must share the same starting point and ending point.
4) An AND-parallel flow, $F_1 \wedge F_2$, indicates that fluids should pass through both F_1 and F_2 at the same time. The flows must share the same starting point and ending point.

In addition to points on the actual flow layer, each chip contains an abstract *source* point and *sink* point which should be considered to reside off-chip. While many intermediate flows may be constructed by the user, only complete flows from the source to the sink are considered as the ISA.

In addition to flows, the language allows one to specify pumping and mixing functionality. The *pump* instruction indicates that fluids should be actively transported along a given flow via placement of peristaltic pumping valves. The *mix* instruction applies pumping to a cyclic flow path, thereby causing the contents of the path to mix. Mixing instructions can be composed in either an AND or OR relationship with other flows and mixers, in order to indicate that mixing only happens in combination with other chip operations. For example, the language can express the fact that a chip either loads a mixer, performs mixing, or drains the mixer. This information is important for minimizing the number of control channels.

Example. Figure 2 provides an example of a flow network. In this chip, the desired operation is to flow fluids from exactly one of the inputs to the output, passing through either the top left or the bottom right of the mixer. The mixer can also be actuated. Using hierarchical annotations, this flow can be expressed as:

$$
\begin{array}{ll}
(source \rightarrow in_1 \rightarrow x \ \vee \ \ldots \ \vee & \\
\ source \rightarrow in_8 \rightarrow x) \rightarrow & \text{[flow from any input]} \\
(x \rightarrow y_1 \rightarrow z \ \vee \ x \rightarrow y_2 \rightarrow z)) \ \vee & \text{[through either side of mixer]} \\
mix(x \rightarrow y_1 \rightarrow y_2 \rightarrow x) & \text{[or actuate the mixer]}
\end{array}
$$

In our tool, the source and sink nodes are indicated implicitly by designating certain points as inputs or outputs (i.e., points that are adjacent to the source or sink). While our tool utilizes a hierarchical description as shown above, this description can also be flattened for the sake of presentation, as shown in the top of Figure 2.

III. Generating Control Logic

After defining an ISA, the next step of the microfluidic designer is to implement a control layer that induces the desired flows. This is accomplished by placing control valves at appropriate chip locations, including inputs, outputs, flow junctions, and pumping and mixing paths.

We describe the first method to automatically generate the control layer, including the placement of valves, the logic of valve operation, the sharing of control lines across valves, and the routing of valves to external control ports. We also generate a graphical user interface (GUI) that allows the user to operate the designed device, invoking each flow with a single mouse click. We establish that certain aspects of control generation are NP-hard, in which case we resort to heuristics that are effective in practice.

A. Problem Definition

We assume that the *source* specified in the flow ISA is pressurized, such that fluids will flow naturally in the direction of the *sink*. Valves must be placed and actuated to prevent the flow from progressing down unintended paths.

A formal description of the problem of generating control logic appears in Problem Definition 1. Given the flow layer and a description of the flow ISA, the problem is to generate control valves that implement the ISA with the minimum number of control channels. Usually there is one valve per control channel; however, valves can also share the same channel if they operate in unison across all of the specified instructions. It is important to minimize the number of control channels because each one requires a separate control port, which requires expensive hardware off-chip and also consumes space on the device.

Problem Definition 1 Generation of Control Logic

Given:

- the flow layer (expressing the chip connectivity)
- the flow ISA (expressing which flows may be active at a given time)

produce:

- placement of control valves sufficient to induce the given flows
- table of control sharing (which valves can share the same control channel, while executing the full ISA)
- table of control logic (which control channels should be pressurized to induce each flow)

while minimizing the number of control channels.

B. Complexity of Control Minimization

The following theorem shows that it is NP-hard to minimize the number of control channels needed to implement an arbitrary flow pattern.

Theorem 1: Given an instance of the control minimization problem (Problem 1) for an ISA with N distinct points, it is NP-hard to decide whether the flows can be invoked with k or fewer control channels.

Proof. We reduce from an instance of the graph coloring problem on a graph with N vertices. For each vertex v, introduce points P_v and Q_v and a flow $F_v = source \rightarrow P_v \rightarrow Q_v \rightarrow sink$. For each edge $e = (v_1, v_2)$, introduce the flow $F_e = F_{v1} \vee F_{v2}$. Set the toplevel ISA to be $F = F_{e1} \wedge ... \wedge F_{em}$, where m denotes the number of edges in the graph.

A solution to the graph coloring problem is valid if and only if for all edges $e = (v_1, v_2)$, the vertices v_1 and v_2 do not share the same color. Likewise, a solution to the constructed control inference problem is valid if an only if for all OR-flows $F_e = F_{v1} \vee F_{v2}$, the alternate flows F_{v1} and F_{v2} do not share the same control line. If such flows did share the same control line, then whenever that control line is activated, it would block both F_{v1} and F_{v2} and thus violate the requirements of F_e. This in turn would violate the ISA, which requires F_e to always be active.

Thus, the control inference problem can be solved with k control lines if and only if the graph coloring problem can be solved with k colors. The NP-hardness of control minimization follows from that of graph coloring. □

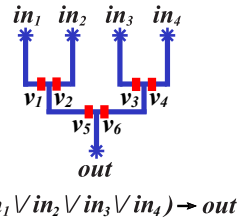
C. Heuristic Solution

Due to the computational complexity of the problem, we resort to heuristics when minimizing the number of control channels in our tool. However, we are still able to guarantee a valid solution, i.e., a placement of valves, control sharing, and control logic that is sufficient to implement the desired flows, whenever such a solution exists.

To make the problem tractable, we make four simplifications:

1) We expand the ISA into a set of directed acyclic graphs, each flowing from the *source* to the *sink*, and exactly one of which is active at a given time. This is accomplished by lifting the disjunctions in OR-parallel flows to the top level of the ISA. This represents an assumption because in the worst case it may not be manageable; nested OR-flows could lead to an exponential expansion of the ISA.
2) We do not consider alternate ways of blocking the inactive half of an OR-flow. (An OR-flow requires that exactly one branch is enabled at once, implying that the inactive branch must be blocked.) In most cases, both branches share a link at the beginning and end, such that enabling one branch automatically disables the other. However, when this is not the case, there may be several combinations of valves that are sufficient to block the inactive path. We choose the blocking valves arbitrarily rather than searching for the combination that minimizes the number of control lines.
3) We reduce the control minimization problem to a graph coloring problem, which itself is NP-hard. We rely on heuristic solutions to graph coloring to enable a heuristic solution to control minimization.
4) We do not attempt to share control lines between pumps or mixers.

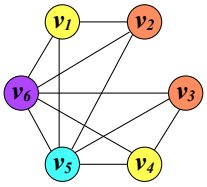
Using these heuristics, our technique for control minimization appears as Algorithm 1 (see Figure 4 for an



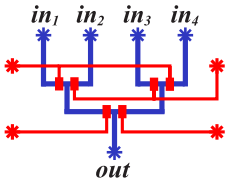
(a) Flow with ISA and valves

	$in_1 \to o$	$in_2 \to o$	$in_3 \to o$	$in_4 \to o$
v_1	open	closed		
v_2	closed	open		
v_3			open	closed
v_4			closed	open
v_5	open	open	closed	closed
v_6	closed	closed	open	open

(b) State table (blank indicates "either")



(c) Graph coloring



(d) Routing with sharing

Fig. 4. Example of control inference. This example highlights the calculation of shared control lines; the valve placement steps are not shown.

Algorithm 1 Heuristic Generation of Control Logic

a) For every point P at the junction of three or more flow segments, place a valve adjacent to P on each segment
b) Expand the ISA into a set of directed acyclic graphs (DAGs), exactly one of which is active at a given time, by lifting OR-flows to the toplevel
c) Build a STATE table that indicates, for each graph G and valve v, whether v needs to be **open**, **closed**, or **either** when G is activated. For each DAG G from Step (2):
   i) For each valve v on the flow layer:
      A) If G passes over v, set STATE(G, v) = **open**
      B) Else if G passes through a point adjacent to v, then set STATE(G, v) = **closed**
      C) Else set STATE(G, v) = **either**
   ii) For all other DAGs $G' \neq G$ in the ISA:
      A) Select a set of valves V such that:
         - closing all valves in V blocks all paths from *source* to *sink* in G'
         - $\forall v \in V$, STATE(G, v) $\neq$ **open**
      B) For each $v \in V$, set STATE(G, v) = **closed**
d) For each valve v: if $\nexists G$ s.t. STATE(G, v) = **closed**, then remove v from the flow layer
e) Map valves to control lines via a reduction to graph coloring:
   i) For each valve v, introduce a vertex Vert(v)
   ii) For each pair of valves (v_1, v_2):
      If $\exists G$ s.t. STATE(G, v_1) $\neq$ **either** and
      STATE(G, v_2) $\neq$ **either** and
      STATE(G, v_1) $\neq$ STATE(G, v_2)
      then introduce edge between Vert(v_1) and Vert(v_2)
   iii) Color the graph with a heuristic graph coloring algorithm, minimizing the number of colors
   iv) Map two valves to the same control line if and only if the corresponding vertices have the same color

example). The algorithm consists of five steps. First, the ISA is expanded into a set of graphs as described previously. Then, valves are conservatively placed at the outlets of every flow junction (any redundancy is removed later). Next, a state table is calculated that determines the state of every valve (open, closed, or either) when each flow graph is active. Subsequently, redundant valves are eliminated from the flow layer; they represent valves that never need to close for the activation of any graph. Finally, the remaining valves are assigned to control lines by reducing the problem to one of graph coloring. Any number of graph coloring heuristics may be used to arrive at an approximate solution (e.g., [18]).

We have also developed control inference algorithms that utilize a looser notion of valve placement. Rather than blocking a flow immediately at a junction, this convention allows valves to be placed anywhere between the junction and the sink. A key benefit of this policy is that it allows multiplexers to be placed across parallel lines. Our tool implements support for loose inference of multiplexers, as demonstrated by example in Figure 2. More details are available in an accompanying report [15].

IV. Routing Control Channels

Following the placement of valves on the control layer, the next step in the microfluidic design process is to connect the valves to external control ports by placing and routing control channels.

We develop the first routing algorithm that is suitable for control channels on microfluidic chips. Our algorithm reduces the problem to one of routing on a grid, with a specific grid size that is dictated by the design rules. Our approach is grounded in an established min-cost max-flow routing algorithm [19], which we extend in two ways to suite the microfluidic context. First, while the previous algorithm produces one line per feature, we introduce the capability for lines to be shared amongst several valves. Our extension requires the use of linear programming to arrive at a solution.

Second, we adjust the produced routes to satisfy aesthetic constraints, such as minimizing the number of corners (changes in direction) of the routes. Such adjustments are critical for the tool to be adopted in practice. Due to space limitations, we refer readers to an accompanying report for details on this step [15].

A. Problem Definition

A formal definition of the routing algorithm appears in Problem Definition 2. The algorithm inputs the layouts of the flow layer, the control ports, and the control valves, as well as the sets of valves that are allowed to share a control line. Optionally, the algorithm also inputs the positions of established control features, such as manually designed or previously-routed lines. The algorithm outputs routes for

Problem Definition 2 Routing of Control Channels

Given:

- placement of flow channels
- placement of p control ports
- placement of n control valves ($n \leq p$)
- permissible sharing of valves amongst m control channels ($m \leq n$)
- (optional) placement of other control features (manually designed or previously routed)

produce:

- routing of control channels from valves to control ports

while respecting:

- sharing constraints
- design constraints

and minimizing the total length of the control lines.

each control channel, thereby connecting each valve to a control port (any control port). The routes must satisfy the sharing constraints: if two valves are connected to the same port, then they must be allowed to share the same line.

The routing is also subject to design constraints, which are summarized in Figure 5. These constraints follow those used in a standard microfluidic foundry [20], [21].

B. Mapping to a Grid

Our algorithm simplifies the general problem formulation into one that operates on a grid. Each valve, control port, and manually-designed control structure is mapped to one or more contiguous points on the grid. The key assumption made by our algorithm is that it is legal to route control lines through all of the unmapped grid points, including points that are adjacent to each other. This assumption places a constraint on the grid size: grid lines must not be any closer than the sum of the control line width and the minimum control line separation. Otherwise, routing different control channels through adjacent grid points would violate the design constraints. (While it would be desirable to allow a separation of more than one grid point between control lines, our attempts to do so led to non-polynomial runtimes.)

There are two approaches to mapping the general design problem into one that operates on a grid. The first approach is to introduce the grid at the beginning of the design process, constraining the designer to place the flow layer along the grid. Using the design parameters from Figure 5, it is possible (with a grid size of 70μm) to arrange flow lines, valves, and control lines such that all features are aligned to the grid while remaining separated by the minimum allowable distance under the design rules. This guarantees that our algorithm finds a feasible and optimal solution (within the space of straight horizontal and vertical control lines) whenever one exists. The second approach is to overlay the grid on a more flexible flow layer, which might include rounded corners and circular mixers that are unaligned with the grid. In this case, we can conservatively approximate the coverage of each feature on the grid, but in so doing we may over-constrain the routing and miss feasible solutions.

Parameter or constraint	**Suggested value**
Width of flow channel	100μm
Width of control channel	30μm
Width of control valve	100μm
Minimum distance between control lines	40μm
Minimum distance between control ports	2000μm
Minimum distance between control port & line	400μm

Fig. 5. Microfluidic design rules.

C. Routing Algorithm

Our routing algorithm follows closely from a min-cost max-flow approach developed for electronic CAD [19] (though our technique requires linear programming). The key innovation in our technique is the ability to share channels between valves that have compatible actuation patterns. Such sharing is critical for reducing the number of control ports, which often limit the scalability of today's microfluidic chips.

Due to space constraints, we omit a complete mathematical formulation of our algorithm. Instead, we provide an intuitive summary of the previous algorithm [19] and then describe our extension.

The min-cost max-flow routing algorithm connects valves to any available control ports in polynomial time [19]. It works by placing the chip on a grid, which it considers as a flow network. Each edge of the grid supports one unit of flow in either direction; points of the grid are modeled as conduit edges that also support one unit of flow. Each valve translates to a source with unit flow, while each control port translates to a sink. If there are pre-existing control features on the chip, then they are treated as obstacles by removing the corresponding grid points from the flow network. Routing is accomplished by maximizing the flow through the network, while minimizing the number of edges utilized (i.e., the cost).

We extend this algorithm in the following way. Rather than modeling a single type of flow over the edges, we introduce multiple flow types, one corresponding to each valve. In the flow graph, each valve represents a unit source of the corresponding flow type. The flow conservation equations are unmodified – flows of each type are independently conserved at all nodes. However, the capacity constraints are modified in an important way: if two valves are allowed to share a control line, then their flows are not both counted against the capacity on an edge. Instead, only the *maximum* flow is counted against the capacity. This allows the control lines from two valves to coalesce without penalty in the case that sharing is permitted.

For example, consider that there are three valves, and the first two valves may share a control line. Then we model three types of flow on each edge. A unit capacity constraint is expressed as follows: $\max(f_1, f_2) + f_3 \leq 1$. In other words, the edge can support either a unit flow for f_3, or a unit flow for *both* f_1 and f_2. This allows the control lines for valves 1 and 2 to both share the edge (if desired). Note that the

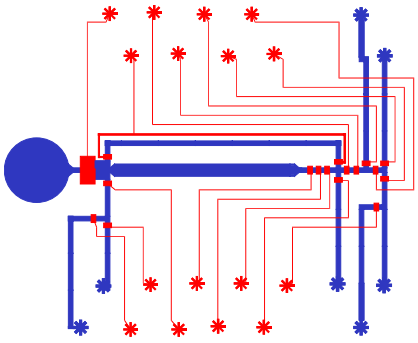


Fig. 6. Cell culture chip with automatic routing.

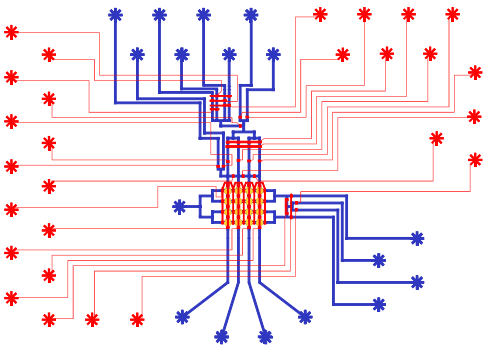


Fig. 7. Waveform generator with automatic routing.

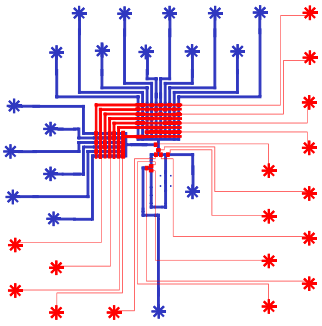


Fig. 8. Metabolite detector with automatic routing.

max function can be implemented as a linear constraint by introducing a new variable f_{max} that is greater than or equal to both f_1 and f_2. Thus, the modified flow network can be solved as a linear program.

While our algorithm permits sharing between specified control lines, it does not enforce that sharing. If it is cheaper to route shared valves to different control ports, then the algorithm will do so. This feature is important, as otherwise the sharing of control lines may introduce undue congestion that makes the routing problem infeasible.

V. Implementation

Micado is a working implementation of our ideas, structured as a plug-in to AutoCAD. Binaries and source code are freely available online [14].

While Micado addresses all of the problems described in this paper (including flow annotations, control inference, and routing), the implementation lags slightly behind the latest algorithms presented here. Minimization of control lines uses a different heuristic, and routing does not connect shared valves at the same time as routing valves to control ports. The exact techniques implemented at the time of this writing are described elsewhere [14], [15].

Micado supports iterative development at each stage. The designer can readily switch back and forth between drawing the flow, annotating the flow, generating the control logic, and routing the channels. Micado also supports the parameterized construction of chips. All of the design rules in Figure 5 are adjustable, including other parameters such as the dimensions of valves and the preferred spacing around flow lines. This allows the designer to easily change the parameters without re-drawing the entire chip.

VI. Evaluation

Micado has been used by at least three researchers as part of their standard microfluidic design flow. We also provided demonstrations and gathered feedback from 12 additional researchers. Designers were particularly excited about the automatic generation of the graphical user interface, which is considerably more usable than their current design flow.

To evaluate the performance of Micado, we focus on three real microfluidic chips that were developed by our collaborators. The first chip (Figure 6) performs a cell culture of a large embryonic cell, with a recirculation loop and sample extraction [22]. The second chip (Figure 7) generates a waveform of temporal stimulants to addressable cell chambers [23]. The third chip (Figure 8) analyzes the metabolites produced by preimplantation embryos [24].

In the figures shown, our routing algorithm is used to connect each valve to a control port. (Valves sharing a control line are connected to each other manually, as the current implementation does not automate this step.) The figures show that our routing algorithm works well in practice, succeeding even in highly dense chips. After a setup phase of 5 seconds or less, all chips are routed in less than 1 second.

While we postpone a formal evaluation of our flow annotation language and control inference algorithm, our design is sufficient to encapsulate the features on the chips examined – with two exceptions. First, on the waveform generator chip, valves are needed to separate cell chambers to prevent diffusion during culture. Since these valves act within a stationary chamber, rather than directing the movement of an active flow, they do not fall into our existing language. Such "separation" primitives could be added in the future, but currently require manual valve placement by the user. The second limitation of our system is manifested elsewhere on the waveform generator chip, where valves are toggled rapidly on neighboring input ports (for example, to draw 30% from one input and 70% from another). While Micado infers the locations of these valves correctly, it is currently unable to express the adjustable and high-frequency switching pattern that is required during operation.

VII. Related Work

Decades of work in electronic CAD [25] can be applied to automate the design of microfluidic chips. However, there are several differences between electronics and microfluidics that will prompt new innovations in CAD tools. First, a current bottleneck to the scalability of microfluidic chips is the cost and complexity of external control ports. Thus, techniques that minimize external ports over other factors are demanded. This difference is reflected in the routing algorithm developed in this paper. To the best of our knowledge, our approach represents the first polynomial-time algorithm that connects each internal feature to any one of the external ports while allowing sharing between specified sets of lines.

A second difference between electronic CAD and microfluidic CAD is the fast turn-around time in manufacturing

microfluidic chips. Because microfluidic chips are easy to fabricate (graduate students go through several fabrication cycles in a week), the design is frequently modified and refined. Thus we need interfaces and abstractions that support iteration, as well as algorithms that are fast enough for real-time usage. A third difference is that microfluidic chips rely on non-rectangular features, such as circular mixers and rounded channels, that do not fit into a grid abstraction. Optimizing the layout of these chips may require reasoning about curves, or may demand very fine or multi-resolution grids to achieve good results.

Prior work in microfluidic CAD targets a different device technology, based on the manipulation of fluids as droplets on a grid [3], [4], [5]. While droplet-based chips are attractive in their flexibility, it is unclear if these architectures can scale to the small volumes offered by multilayer soft lithography due to surface energy considerations. Chakrabarty and Zeng provide an overview of design automation for droplet-based microfluidics [11]. As droplet-based chips can route fluids in any pattern on a grid, the CAD problem addresses how to map a given experiment to the chip, rather than designing the chip itself. Thus, our algorithms for automating the design of the control layer do not have an immediate analog in droplet-based microfluidics. Nonetheless, our routing algorithm could be applied to place "virtual channels" for fluids on a droplet grid, in the context of a given experiment.

Previously, our group [17] and others [16], [26] have researched high-level programming abstractions to enable users to easily write complex experiments for a given chip. While these efforts also included a notion of a microfluidic ISA, the ISA was designed for an existing chip, rather than serving as a specification for the control logic of a new chip.

VIII. Conclusions

This paper formulates two new problems for microfluidics CAD, and proposes the first known solution to each. The first problem is the inference of control logic for a given instruction set. We prove that minimizing the number of control channels is NP-hard and offer a heuristic algorithm that works well in practice. We also provide the first interface for hierarchically specifying the instruction set of interest. The second problem is the routing of control channels from valves to control ports, while allowing sharing between certain sets of channels. We adapt an algorithm from electronic CAD, deriving an appropriate grid size and extending it to allow sharing, in order to arrive at a good solution.

We implemented a version of our algorithms [15] in an AutoCAD plug-in, called Micado, which we have released to the research community. To our knowledge, this tool is the first to automate the design of multilayer microfluidic chips. While such design tools have been in development for droplet-based microfluidic processors, our work is the first to target the broad class of soft-lithography chips. Such automation will be critical if we are to fully exploit the potential of building scalable and programmable microfluidic chips with millions of independent parts.

IX. Acknowledgments

We are very grateful to J.P. Urbanski and David Craig, who provided extensive guidance and support. We also thank Todd Thorsen, Jessica Melin, Stephen Quake (and the members of his group), Moeto Nagai, Filippo Menolascina, and Ahmad Khalil for helpful feedback. This work was supported by the National Science Foundation (# CCF-0541319).

References

[1] T. M. Squires and S. Quake, "Microfluidics: Fluid physics at the nanoliter scale," *Reviews of Modern Physics*, vol. 77, 2005.

[2] G. M. Whitesides, "The origins and the future of microfluidics," *Nature*, vol. 442, pp. 368–373, 2006.

[3] M. G. Pollack, R. B. Fair, and A. D. Shenderov, "Electrowetting-based actuation of liquid droplets for microfluidic applications," *Applied Physics Letters*, vol. 77, no. 11, pp. 1725–1726, 2000.

[4] S. K. Cho, H. Moon, and C.-J. Kim, "Creating, transporting, cutting, and merging liquid droplets by electrowetting-based actuation for digital microfluidic circuits," *Journal of Microelectromechanical Systems*, vol. 12, no. 1, 2003.

[5] P. R. C. Gascoyne et. al, "Dielectrophoresis-based programmable fluidic processors," *Lab on a Chip*, vol. 4, pp. 299–309, 2004.

[6] M. A. Unger, H.-P. Chou, T. Thorsen, A. Scherer, and S. R. Quake, "Monolithic microfabricated valves and pumps by multilayer soft lithography," *Science*, vol. 288, no. 5463, pp. 113–116, April 2000.

[7] G. M. Whitesides, E. Ostuni, S. Takayama, X. Jiang, and D. E. Ingber, "Soft lithography in biology and biochemistry," *Annual Review of Biomedical Engineering*, vol. 3, pp. 335–373, 2001.

[8] J. Hong and S. Quake, "Integrated nanoliter systems," *Nature BioTechnology*, vol. 21, no. 10, pp. 1179–1183, 2003.

[9] "Fluidigm 96.96 dynamic array," http://www.fluidigm.com/pdf/fldm/FLDM_MRKT00110.pdf.

[10] J. Perkel, "Microfluidics: Bringing new things to life science," *Science*, Nov 2008.

[11] K. Chakrabarty and J. Zeng, "Design automation for microfluidics-based biochips," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 1, no. 3, 2005.

[12] K. Böhringer, "Modeling and controlling parallel tasks in droplet-based microfluidic systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 2, 2006.

[13] F. Su and J. Zeng, "Computer-aided design and test for digital microfluidics," *IEEE Design and Test of Comp.*, vol. 24, no. 1, 2007.

[14] "Micado website," http://groups.csail.mit.edu/cag/micado/.

[15] N. Amin, "Computer-aided design for multilayer microfluidic chips," M.Eng Thesis, Massachusetts Institute of Technology, 2008.

[16] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson, "AquaCore: A programmable architecture for microfluidics," in *ISCA*, 2007.

[17] W. Thies, J. P. Urbanski, T. Thorsen, and S. Amarainsghe, "Abstraction layers for scalable microfluidic biocomputing," *Nat. Comp.*, May 2007.

[18] D. Karger, R. Motwani, and M. Sudan, "Approximate graph coloring by semidefinite programming," *J. of the ACM*, vol. 45, no. 2, 1998.

[19] H. Xiang, X. Tang, and D. F. Wong, "Min-cost flow-based algorithm for simultaneous pin assignment and routing," *IEEE Trans. on Computer-Aided Design of Integ. Circ. and Sys.*, vol. 22, no. 7, 2003.

[20] "Stanford microfluidic foundry: Basic design rules," http://thebigone.stanford.edu/foundry/services/basic_design_rules.html.

[21] J. Melin and S. Quake, "Microfluidic large-scale integration: The evolution of design rules for biological automation," *Annual Reviews in Biophysics and Biomolecular Structure*, vol. 36, pp. 213–31, 2007.

[22] J. P. Urbanski, "Microfluidic tools for metabolomics," Ph.D. dissertation, Massachusetts Institute of Technology, September 2008.

[23] N. Andrew, D. Craig, J. P. Urbanski, J. Gunawardena, and T. Thorsen, "Microfluidic temporal cell stimulation," in *MicroTAS*, 2008.

[24] J. P. Urbanski, M. T. Johnson, D. D. Craig, D. L. Potter, D. K. Gardner, and T. Thorsen, "Noninvasive metabolic profiling using microfluidics for analysis of single preimplantation embryos," *Analytical Chemistry*, vol. 80, no. 17, 2008.

[25] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. New York: John Wiley and Sons, 1990.

[26] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson, "Automatic volume management for programmable microfluidics," in *PLDI*, 2008.