UC Riverside

UC Riverside Electronic Theses and Dissertations

Title

Design Automation of Continuous Flow-Based Microfluidic Biochips

Permalink

https://escholarship.org/uc/item/5cc8573p

Author

McDaniel, Jeffrey

Publication Date

2016

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, availalbe at https://creativecommons.org/licenses/by/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Design Automation of Continuous Flow-Based Microfluidic Biochips

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Jeffrey Michael McDaniel

August 2016

Dissertation Committee:

Professor Philip Brisk, Chairperson
Professor Frank Vahid
Professor Daniel T. Grissom
Professor William H. Grover

Copyright by
Jeffrey Michael McDaniel
2016

The Dissertation of Jeffrey Michael McDaniel is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to thank my advisor, Dr. Philip Brisk, without his help I would not be here. I began working under Dr. Brisk as an undergraduate, during which time he helped me to find a project that I was truly passionate about. Throughout my graduate career he has been supportive, not just of my research but the many extracurricular activities I have undertaken. His wealth of knowledge has been instrumental in me growing as a researcher, a professional, and a person.

The following works have been previously published and have contributed significantly to this dissertation.

- McDaniel, J., Parker, B., & Brisk, P. (2014). Simulated annealing-based placement for microfluidic large scale integration (mLSI) chips. 2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC), 4, 16. http://doi.org/10.1109/VLSI-SoC.2014.7004170

- McDaniel, J., Crites, B., Brisk, P., & Grover, W. H. (2015). Flow-layer physical design for microchips based on monolithic membrane valves. IEEE Design and Test, 32(6), 5159. http://doi.org/10.1109/MDAT.2015.2459699

- McDaniel, J., Curtis, C., & Brisk, P. (2013). Automatic synthesis of microfluidic large scale integration chips from a domain-specific language. In 2013 IEEE Biomedical Circuits and Systems Conference, BioCAS 2013 (pp. 101104). http://doi.org/10.1109/BioCAS.2013.6679649

- McDaniel, J., Crites, B., Curtis, C., & Brisk, P. (2015). Design Automation for Flow-based Microfluidic Biochips. In 37th Annual International conference of the IEEE Engineering in Medicine and Biology Society.

- McDaniel, J., & Baez, A. (2013). Design and Verification Tools for Continuous Fluid Flow- based Microfluidic Devices. In Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC (pp. 219224).

I would like to thank the following entities for their generous awards, fellowships and stipends which made this research possible.

- UCR for their initial Dean's fellowship aid and the Dissertation Year Fellowship.

This dissertation is dedicated to John and Janet McDaniel, my supportive and understanding parents. They remained encouraging throughout all the moments my final dissertation seemed to be getting further and further away.

ABSTRACT OF THE DISSERTATION

Design Automation of Continuous Flow-Based Microfluidic Biochips

by

Jeffrey Michael McDaniel

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, August 2016
Professor Philip Brisk, Chairperson

Continuous flow-based microfluidic biochips use actively controlled valves to manipulate the flow of fluids through components and channels to automate biochemical protocols. The current state-of-the-art design practices require biochemists with no engineering experience to manually design these devices. This process severely limits the realistic complexity a device can have, limiting the applications they can be used for. These devices can be viewed as a netlist of components and their interconnections, along with a protocol to be performed, similar to a how an integrated circuit can be abstracted. This thesis introduces an end to end tool-chain capable of taking a high level biochemical protocol and automatically generating a device design that can be fabricated to perform that experiment. This is done through the use of design automation algorithms adapted from the electronic design automation (EDA) industry.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Microfluidics-based biochips have become an actively researched area in recent years. These devices, known as laboratory-on-a-chip (LoC) or microfluidic total analysis systems (μTAS), manipulate fluids at the micro- or nano-liter scale to automate biochemical experiments. These devices fall into several different categories, digital microfluidic biochips (DMFB), paper-based microfluidic devices, and continuous flow-based microfluidic devices. The focus of this thesis is on continuous flow-based microfluidic devices.

1.1 Continuous Flow-Based Microfluidic Devices

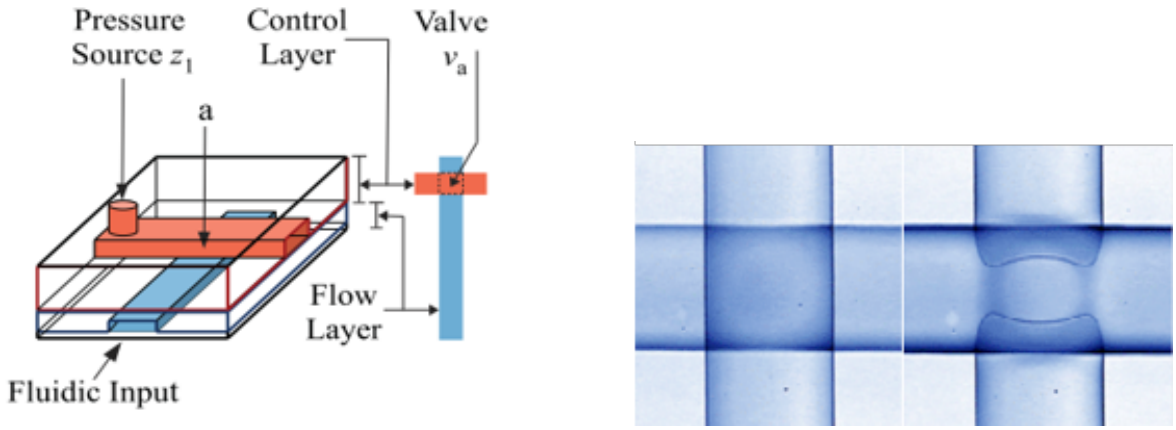
In the continuous flow paradigm, as the name suggests, the device is made up of channels through which the samples continuously flow. This can be particularly useful in cell culturing where a medium needs to be flow over the cells to provide nutrients for growth in a controlled manner. These devices were originally passive, fluid would be either flowing through the entire device or not flowing at all, but With the advent of microfabrication techniques researchers were able to integrate active valving to control the flow of fluids. This enabled the development of biochemical components (e.g., dispensers, filters, mixers, separators, detectors)

which could be integrated on the device to perform more complex biochemical experiments [80]. These microsystems offer several advantages over the conventional biochemical analyzers, e.g., reduced sample and reagent volumes, reduced reaction time, ultra-sensitive detection and higher system throughput with several assays being integrated on the same chip [86].

As the valving technology improved these devices became known as "microfluidic Very Large Scale Integration" (mVLSI) devices [13]. The chip has two logical layer types: *flow layer* and the *control layer*. The liquid in the flow layer is manipulated using the control layer to actuate valves. Theoretically there could be multiple flow and control layers, however, to date, the most complex chip utilizes only three layers (two control sandwiching one flow layer) [14]. These devices are typically fabricated using a flexible elastomer such as Polydimethyl Siloxane (PDMS) to create the valve. The two main technologies for these valves include the microfabricated valves developed at Stanford University by Dr. Stephen R. Quake [83] and the monolothic membrane valves developed at Berkeley under Dr. Richard A. Mathies [35].

The microfabricated valves use multilayer soft lithography to fabricate the layers separately then aligning and bonding these layers using alignment marks. This method allows for rapid prototyping, and retains the biocompatibility properties of the PDMS. The valves are create by having channels on the control layer cross the channels along the flow layer, as shown in Fig. 1.1. When the control channel is pressurized, the channel expands and the thin layer of PDMS between the flow and control channels flexes to block the flow channel, closing the valve.

The monolithic membrane valves use chemical etching to create channels on two layers of glass, which become the "flow" and "control" layers. The two layers of glass are aligned and then a thin layer of PDMS is then bonded between them to create the valves as shown in Fig. 1.2. In this paradigm, a vacuum is applied to the control channel, drawing the PDMS



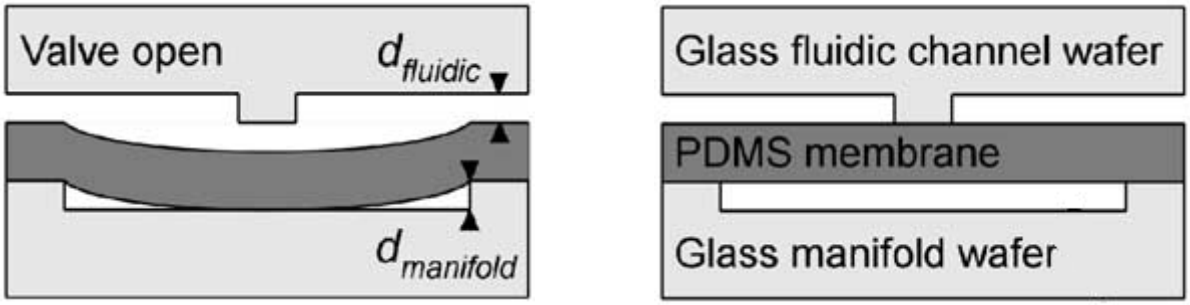
(a) (b)

Figure 1.1: (a) The microfabricated valve from Stanford University. The bottom layer (blue) indicates the flow layer and the top layer (red) is the control layer. (b) When the control channel is pressurized the PDMS flexes and closes the flow channel, blocking the flow of fluids.

into the control channel, allowing the fluid to flow through the flow channel, opening the valve.

The fabrication methodologies presented above are more suited for rapid prototyping or devices produced in small quantities, such as those used in research applications. The processes do not scale up to mass production quantities that are needed in fields such as point-of-care-diagnostics. These devices need to be fabricated using injection molding typically, which leads to more complex valve technologies.

In academia, LoC devices have been used most prominently in the biomedical field for high throughput biochemistry like drug discovery, DNA analysis, or other research applications [26, 37, 38, 40, 55, 59, 80]. However these devices have been gaining more traction in industry for immediate point-of-care-diagnostics [19, 20, 28],food control testing [93], environ-



(a) (b)

Figure 1.2: (a) The monolithic membrane valve naturally closed and (b) when a vacuum is applied to the control channel the membrane flexes, opening the valve.

mental monitoring [23, 48] (e.g., air and water samples) and biological weapons detection [30]. The following are some prominent example applications for flow-based LoC devices:

- *Drug Discovery:* LoC devices allow massively–parallel, high throughput testing of molecules, which is ideal for drug discovery. For example, in order for the hepatitis C virus to proliferate one of its proteins needs to interact and bind with the RNA (Ribonucleic acid). The flow-based chip in [26] has been used to screen over 1,200 small molecules to test if such a protein-RNA interaction was inhibited and 14 such molecules were found. The results were later used to develop a drug which is now in clinical trials.

- *Diagnostic Testing:* The biochip in [19] has been designed for testing HIV and syphilis. The chip is cheap, easy to use, requires only micro-litres of the blood sample and it simultaneously tests for HIV and syphilis giving out the result within 20 minutes. The chip has been utilized successfully in Rwanda to test hundreds of locally collected human samples. In this chip, microfluidic procedures of fluid handling and signal detection have been integrated into a single, easy to use, point-of-care device that replicates all the steps

of the current state-of-the-art, at a lower material cost and without trained interpretation of the results [19].

- Prenatal Screening: In [20], proof of concept studies for a chip that can be used for non-invasive prenatal test to test for chromosomal abnormalities have been reported. The mothers blood is used to measure the fetal DNA (Deoxyribonucleic acid). This chip has been used to successfully identify cases of trisomy 21 (Down syndrome), trisomy 18 (Edward syndrome) and trisomy 13 (Patau syndrome) [20]. A company, Verinata Health , was launched earlier this year to make this technology available to general public and has since been acquired by Illumina [5].

Chapter 2

Related Works

During the last decade, a significant amount of work has been carried out on the individual LoC components as well as the entire platforms [25, 57, 60]. The manufacturing technologies used for the flow-based biochips has advanced faster than Moores law [39] in terms of how many valves can be integrated on a single device. Although these devices are becoming more complex everyday, Computer Aided Design (CAD) tools are still in their infancy. Initial CAD research has been focused on device-level physical modeling of components [77, 52] where designers are using full-custom and bottom-up methodologies involving many manual steps to implement these chips.

Currently, researchers manually map the applications to the valves of the chip using some custom interface (analogous to exposure of gate-level details in microelectronics) [79]. The manual process is quite tedious and needs to be repeated every time a change is made either to the chip architecture or the biochemical application. For larger chips and applications, the process can easily result in inefficient architectural mappings. As the chips grow more complex (commercial biochips are available which use more than 25,000 valves and about a million features to run 9,216 polymerase chain reactions in parallel [71]) and the need of having

multiple concurrent assays on the chip becomes more significant, these methodologies become highly inadequate. Therefore, new top-down design methodologies and design tools are needed, in order to provide the same level of CAD support to the biochip designer as is currently taken for granted in the semiconductor industry. The design process can take months or years for a device which would be considered simple conceptually in the semiconductor industry.

The LoC device can be abstractly viewed as a netlist of components and the interconnections between them, akin to the way an application specific integrated circuit (ASIC) is viewed. In this paradigm the adders, multipliers and memory units are replaced with mixers, heaters and storage units, and the wires are replaced with channels containing volume. Their are many similarities, however, and researchers have proposed significant work on top-down synthesis methodologies for microfluidic devices that utilize algorithms from the semiconductor industry and adapt them to the new constraints.

The system level specifications of the device (e.g., chip area, fabrication technology) and the application (e.g., dependency graph) are taken as the input. The mVLSI design flow starts with generating a **schematic design** of the biochip. Minhass et. al proposed a topology graph-based system-level model of a biochip architecture, independent of the underlying biochip implementation technology in Ref. [66]. This is followed by the **physical design** of the flow layer, consisting of placement of the components [68, 63, 82, 72] and routing of flow channels [68, 58, 41] while following the design rules. Researchers have also proposed integrated placement and routing approaches for the flow layer [68, 72]. The routing latencies for the flow fluids between operations can now be estimated using the channel lengths and geometries and specific fluid properties (e.g., viscosity). These latencies, and operation time, are used for **application mapping** of the given biochemical application onto the biochip architecture and the optimized

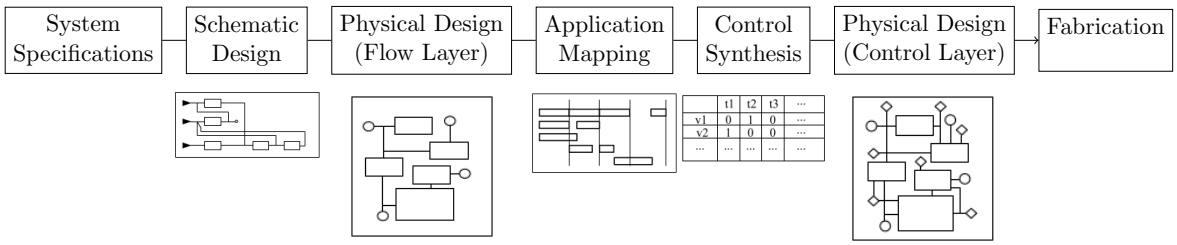


Figure 2.1: The mVLSI design flow from a system specification a schematic design is generated. The physical design of the flow layer is then placed and routed and the application is mapped to the architecture. The control layer is synthesized, and optionally optimized, before the control layer is routed. Finally the device is ready for fabrication.

schedule for its execution is generated. Application mapping and scheduling have been proposed in [66, 82, 24, 67].

If the desired application requires only a passive device to be completed then a fabrication specification can be generated at this point. If active control is required, then the control information (which valves to open and close at what time and for how long) can now be extracted. The **control synthesis** step uses optimization schemes [68, 42, 92] to minimize the external pressure sources required by the control layer, reducing the macro-assembly around the chip. This is followed by the control layer routing [41, 42]. The finalized device is then ready to be sent to fabrication. The fabrication process may introduce defects, which could lead to costly application failure (due to high reagent costs and hard-to-obtain samples). Hence, testing techniques [43] have been proposed to screen the chips after fabrication. This process is typically iterated over several times before a reliable architecture can be fabricated.

Chapter 3

Overview

Figure 3.1 shows the tool-chain flow described in this thesis. The flow is split into several sections separated by the dashed boxes; the high level language compilation on top, followed by device assembly, programming, and simulation on the bottom left, and finally the physical synthesis on the bottom right. The top level input to the system is a high level language **assay specification** of the experiment to be run. The BioCoder language [11] was extended in Ref. [62] to apply to LoC devices and their constraints. The specification is then compiled, generating an intermediate representation (IR) in a virtualized assembly language [62] and javascript object notation (JSON). The IR encodes a **sequencing graph** of the operations to be performed and their dependencies in the form of a directed acyclic graph (DAG). This sequencing graph, combined with the available **entity libraries** is input into the **architecture synthesis** step to allocate the necessary components and specify their interconnections. This generates an **LoC architecture specification** in the microfluidic Hardware Design Language (mHDL). *Note: If the user wishes to describe a device architecture without a specific experiment in mind the mHDL language can be used and the previous steps can be skipped.*

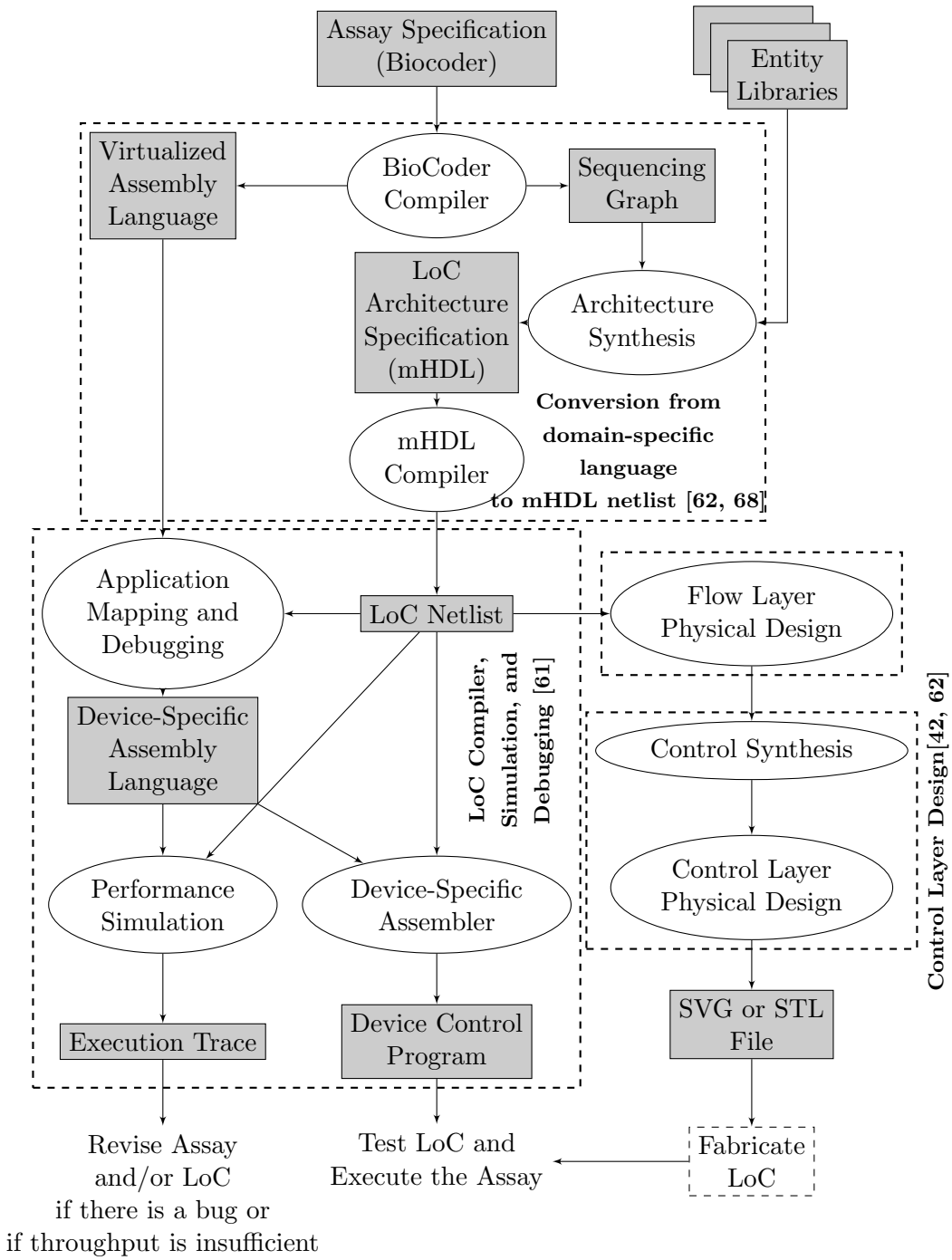


Figure 3.1: Full flow of the tool-chain from a high level language description of the experiment to be run to a fabrication specification, device control program and simulation output of the execution.

The LoC device can be abstractly viewed as a **LoC Netlist** of components and their interconnections. This abstract view is encoded in a JSON formatted IR which can be generated from the **mHDL Compiler**. This netlist is fed into the **application mapping and debugging** stage which performs the scheduling of the application onto the architecture. The scheduling of the application produces a **device-specific assembly language** [62] which can be used to generate the **device control program** through the **device specific assembler** or used in the **performance simulation** to generate an **execution trace** of the application being performed on the given architecture. The execution trace can be used to iterate through designs before fabricating the final design, a much more expensive process.

The LoC Netlist is also used to initiate the physical synthesis process, shown on the bottom right. This process begins with the placement and routing of the flow layer(s) in the **flow layer physical design** step. The control valves are then deterministically placed on the device in the **control synthesis** step. The control valves can optionally be optimized during this step to reduce the number of external pressure sources needed to drive the device. The **control layer physical design** consists of placing the external pressure sources on the device and routing the corresponding channels between each pressure source and the control valve(s) it drives. Finally the design is completed and the last step is to generate a file that can be used to fabricate the device. A scalable vector graphic (SVG) file can be used to generate the masks necessary for the soft lithography and chemical etching processes used for the microfabricated valve and monolithic membrane valve technologies (respectively). A stereolithograpy (STL) file can be generated as well if the desired fabrication technology is injection molding, used most commonly in industry, or 3D printing, used for rapid prototyping typically.

Chapter 4

Preliminaries

4.1 Models

4.1.1 Architecture Model

Figure 4.1 shows the functional view of a flow-based biochip with five inputs, five outputs, two mixers, one heater, one filter and one detector. Physically, the biochip can have multiple layers, but the layers are logically divided into two types: flow layer(s) and control layer(s). The liquid in the flow layer is manipulated using the control layer.

The architecture is represented as a device netlist, or graph, $D = (V, C)$ where V is the set of vertices, or nodes, and C is the set of connections with c_{ij} being a connection between v_i and $v_j \in V$. The set V is made up of two types of nodes, $S \subset V$ representing the switches in the device and $M \subset V$ representing the non-switch components. S and M are disjoint sets such that $S \cup M = V$. In Fig. 4.1, $V = \{S, M : s_1 \dots s_{11} \in S$ and $In_1 \dots In_5, Out_1 \dots Out_5, Mixer_1, Mixer_2, Det_1, Heater_1,$ and $Filter_1$ are the $m \in M\}$. The connection between In_1andS_1 can be represented by $c_{In_1,s_1} \in C$.

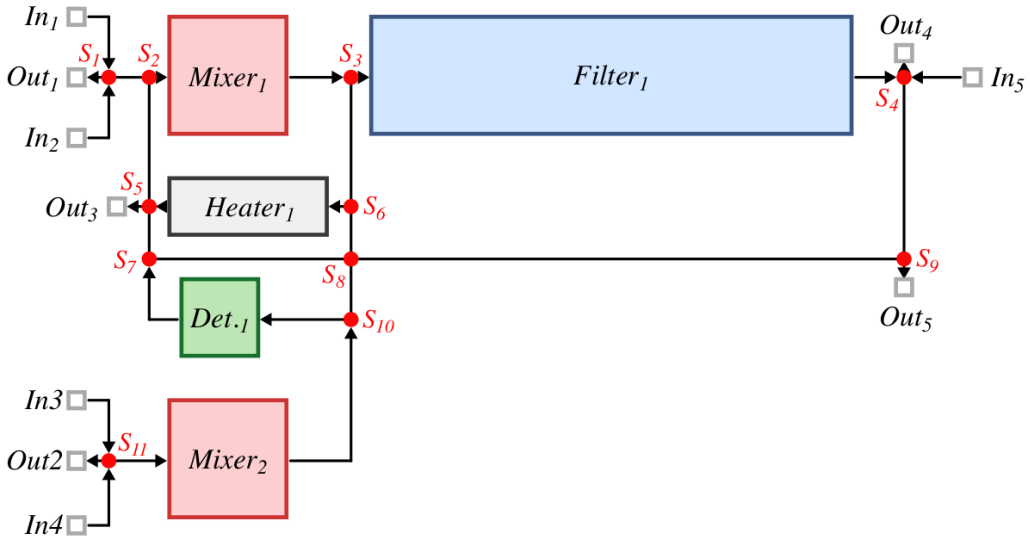


Figure 4.1: Functional view of an LoC with five inputs, five outputs, two mixers, one heater, one filter, one storage container and eleven switches.

This model of the given architecture allows us to abstract away the finer details and focus on the overall design of the device. The nodes in this graph representation of the architecture, however, are not individual points but microfluidic components with positive dimensions. Akin to the semiconductor where gates are made of transistors, and more complex units are made up of these gates, the nodes in the architecture model represent more complex units, *entity types* that are made up of the basic building block of the LoC, the valve. In Fig. 4.1 $Mixer_1$ and $Mixer_2$ are both of type $Mixer$, Fig. 4.3. To make the system easier to use and more extensible an entity model $E = (F, H)$ is introduced where F is the set of functions the entity is capable of performing and H is the dimensions of a bounding box for the entity defined as $H := width \times height \times depth$ or $w \times h \times d$. *Note: The architecture model can be viewed as a set of layers, each of which represented in 2D space. In this model the depth of the entities is ignored until generating the final fabrication specification.*

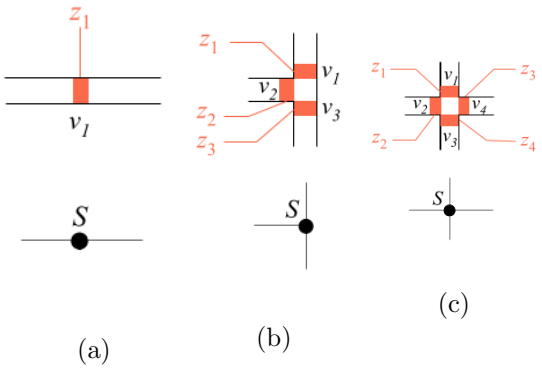


Figure 4.2: Switch configurations with (a) two directions (b) three directions and (c) four directions.

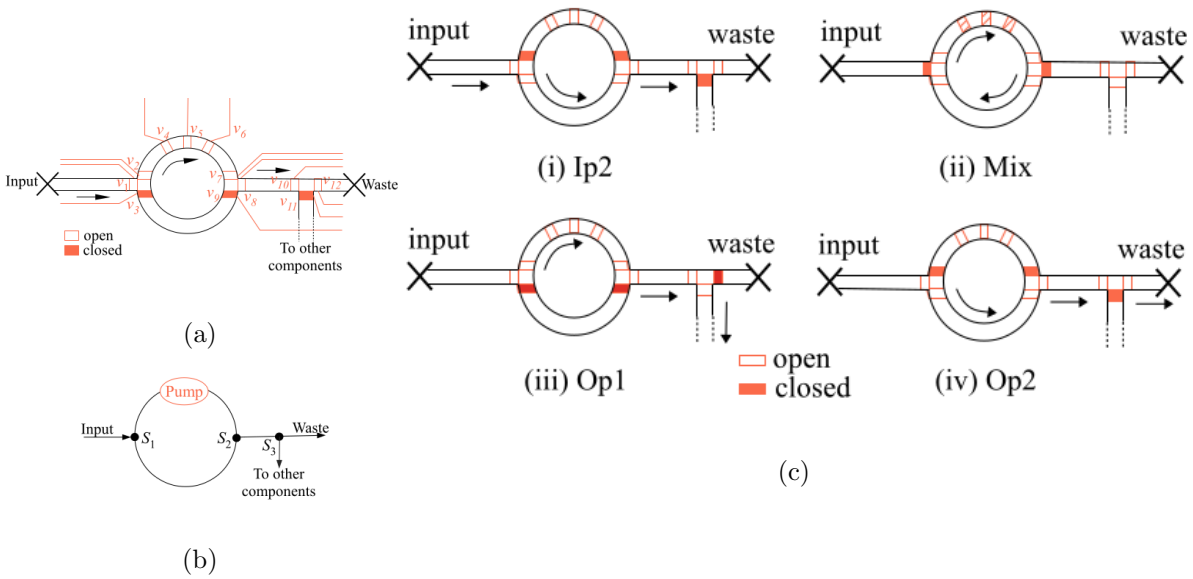


Figure 4.3: (a) Schematic view of a mixer component (b) conceptual view of the mixer and (c) operational phases of the mixer

The *Mixer* entity type has one function $f \in F$ (mix) which is defined by its five operational phases shown in Figs. 4.3a and 4.3c. The first two phases represent the input of two fluid samples that need to be mixed, followed by the mixing phase. The mixed sample is then transported out of the mixer in the last two phases. For the first fluidic input (phase Ip1, see Fig. 4.3a), valves v_1, v_2, v_7 and v_8 are opened (together with v_4, v_5, v_6), the pump at the *Input* is activated and the liquid fills in the mixer upper half.

In the next phase Ip2, the second fluid sample fills the lower half of the mixer (Fig. 4.3c-i). Once both halves are filled, the mixer input and output valves ($v_1 and v_8$) are closed while valves v_2, v_3, v_7, v_9 are opened and the mixing operation is initiated (Fig. 4.3c-ii). Valve set $\{v_4, v_5, v_6\}$ acts as a peristaltic pump. Closing valve v_4 inserts some pressure on the fluid inside the mixer, closing valve v_5 creates further pressure, then as valve v_6 is closed valve v_4 is opened again. This forces the liquid to rotate clockwise in the mixer. The valves are closed and opened in a sequence such that the liquid rotates at a certain speed accomplishing the mixing operation. Next, in phase Op1 (Fig. 4.3c-iii), half of the mixed sample is pushed out of the mixer towards the rest of the chip and in Op2 (Fig. 4.3c-iv), the other half is transported to the waste.

Table 4.1 shows the entity library $\mathcal{L} = E(F, H)$ of six commonly utilized microfluidic components [10]. The geometrical dimensions H are given as lengthwidth and are scaled, with a unit length being equal to 150 µm, i.e., a length of 10 in Table 4.1 corresponds to 1500 µm. The different operational phases listed for a component may or may not be executable in parallel depending on how the component is implemented. E.g., the mixer presented here has only one input port to receive both input fluids, thus only one input phase can be activated at a time.

The individual components of the LoC device are represented using the component model $m = (e, x, y)$ where (x, y) is the location of the top left corner of the component on the layer and $e \in E$ is the entity type of the given component. The size and functionality of the

Table 4.1: Entity Library ($\mathcal{L}$)

Entity	**Operational Phases**	**Exec. Time**	**H**
Mixer	Ip1/Ip2/**Mix**/Op1/Op2	0.5 s	30×30
Filter	Ip/**Filter**/Op	20 s	120×30
Detector	Ip/**Detect**/Op	5 s	20×20
Separator	Ip1/Ip2/**Separate**/Op1/Op2	140 s	70×20
Heater	Ip/**Heat**/Op	20 °C/s	40×15
Storage	Ip or Op	–	90×30

component can be extracted from the entity type. The component model m is also assumed for switches $s \in S$ where the entity type e is a switch in the fabrication technology.

Transporting a fluid through the flow channels requires (i) the fluid that needs to be transported; this fluid is already inside the chip as the output of an operation, or it has to be dispensed from a fluid reservoir connected to an input port, such as In_1 in Fig. 4.1, (ii) a pressure source that can move the fluid, typically obtained with an off-chip pump; alternatively it can be generated on-chip by a peristaltic micropump and, (iii) an output port to remove the displaced fluid. *Note: The displaced fluids can remain on-chip but this would lead to the problem of "dead fluid management", i.e., moving displaced fluids around, such that they do not interfere with the chip operation.*

In an active LoC the valves are represented on the flow and control layer as they indicate a crossing point between a flow layer channel and a control layer channel. These valves are used to control the transportation of fluids through the device. The control layer model captures the valve actuation details required for the on-chip execution of all operational phases

of an entity. Table 4.2 presents the control layer model of the pneumatic mixer presented in Fig. 4.3a. In Table 4.2, the valve activation for each phase is shown, 0 representing an open and 1 a closed valve. The status Mix shown for the valve set $\{v_4, v_5, v_6\}$ on row 4 of Table 4.2 represents the mixing step in which these valves are opened and closed in a specific sequence to achieve mixing. LoC devices have many methods of control, however these methods all require the same information, represented in the actuation sequences of the control layer model.

The paths the fluids take through the devices are defined as a set of routes R where each route $r_i \in R$ is a set of components $m \in M$ and connections $c \in C$ in the architecture. In Fig. 4.1, $r(In_1, Mixer_1) = \{In_1, c_{In_1,s_1}, s_1, c_{s_1,s_2}, s_2, c_{s_2,Mixer_1}, Mixer_1\}$ is the route from In_1 to $Mixer_1$. The entire flow path is occupied, and unusable, until the completion of the fluid transportation. This imposes routing constraints on the device; two routes are mutually exclusive if they share any component or connection along their respective routes, i.e. routes r_1 and r_2 are mutually exclusive if $\exists v_i \in V$ or $c_{ij} \in C$ such that $v_i \in r_1$ and r_2 or $c_{ij} \in r_1$ and r_2. If a route flows through a channel c_{ij} it will by definition flow through v_i and v_j and so the above can be simplified to only look at all $v \in r_1$ and r_2. Let $r_1 = r(In_1, Mixer_1)$ from above and $r_2(Heater_1, Mixer_1) = \{In_1, c_{In_1,s_1}, s_1, c_{s_1,s_2}, s_2, c_{s_2,Mixer_1}, Mixer_1\}$, r_1 and r_2 are mutually exclusive because they share s_2 and $Mixer_1$.

Now with the routes defined, the time it takes for the transportation of a fluid, unlike in semiconductors, is non-trivial. The latency of a connection is defined with $l(c_{ij})$ and the latency of an entire route with $l(r_i) = \sum_{c_{ij} \in r_i} l(c_{ij})$. It is important to note here that the routing latency can only be known after the device is fully placed and routed. Prior to placement and routing, an estimate for the routing latency can be used.

Table 4.2: Mixer: Control Layer Model

Phase	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
1. Ip1	0	0	1	0	0	0	0	0	1
2. Ip2	0	1	0	0	0	0	1	0	0
3. Mix	1	0	0	Mix	Mix	Mix	0	1	0
4. Op1	0	0	1	0	0	0	0	0	1
5. Op2	0	1	0	0	0	0	1	0	0

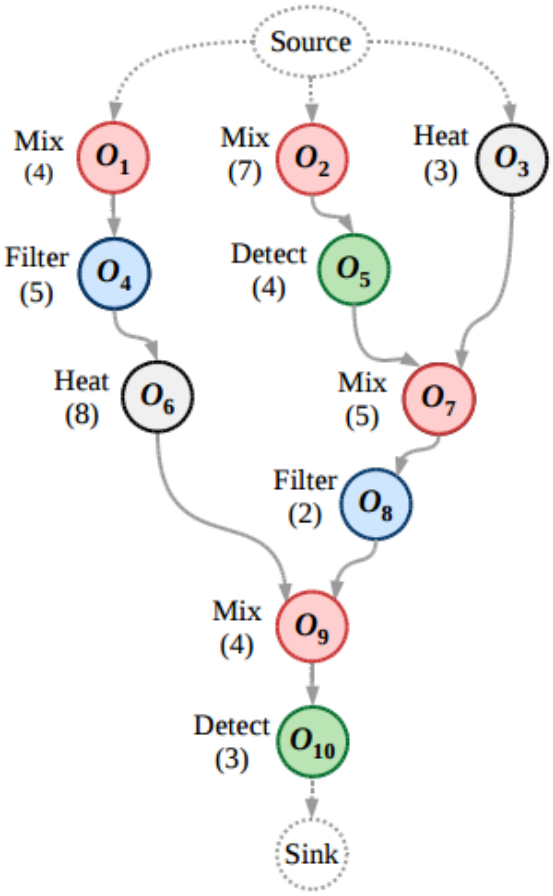


Figure 4.4: Sequencing graph

4.1.2 Application Model

The application model of the biochemical experiment, or assay, is represented as a sequencing graph, shown in Fig. 4.4, $\mathcal{G} = (Op, \mathcal{E})$. The sequencing graph $\mathcal{G}$ is a directed and acyclic graph (DAG) which is polar, i.e., there is a source vertex that has no predecessors and a sink vertex that has no successors. Figure 4.4 shows an example of a biochemical application model with four mixing operations (op_1, op_2, op_7, op_9), two filtering operations (op_4, op_8), two heating operations (op_3, op_6), and two detecting operations (op_5, op_{10}). The execution times for the operations are given in parentheses below the operation node.

The application model is bound to a specific device architecture model using a binding function $\mathcal{B} : Op \rightarrow \mathcal{M}$. Each vertex (operation $op_i \in Op$) is bound to a component $m \in M$ of the architecture D. The switches in the architecture do not execute operations, rather they are used for transporting fluids. The operations of the biochemical experiment have an execution time represented with $t(op_i)$. This execution time is only an estimate of the execution time as different components may complete the same operation in a different amount of time. A more precise representation of the execution time, after binding, is given by $t(op_i, m_j)$, which indicates the execution time of a given operation op_i on the the component m_j.

The edge set, $\mathcal{E}$, models the dependency constraints in the assay, i.e., an edge $e_{i,j} \in \mathcal{E}$ from op_i to op_j indicates that op_j is *dependent* on op_i. The fluid output of op_i must be used as an input for op_j and so it must be transported from $\mathcal{B}(op_i) = m_i$ to $\mathcal{B}(op_j) = m_j$. If the fluid output from op_i cannot be used immediately by op_j (e.g., it has to wait for another operation to finish on component m_j), it has to be stored in a "storage unit", see Table 4.1. Each storage unit is capable of storing one or more fluid unit samples, depending on the number of storage channels inside the unit. Operations have one incoming edge for each input required. These inputs correspond to the input phases (Ipx) of the corresponding component, e.g., a mixing

operation has two incoming edges because the mixer component has two input phases (Ip1 and Ip2). Similarly, an operation has at most one outgoing edge for each output phase of the corresponding component. If it has fewer outgoing edges than the number of output phases it is assumed that the remaining fluids are transported to the waste.

Part II

High Level Synthesis

Chapter 5

High Level Languages

5.1 Introduction

In previous research, the top level input to the system is a sequencing graph representation of the assay to be executed. These sequencing graphs, however, lacked standardization in how they were specified and the burden lay on the "programmer" to describe the sequencing graph. In this case, the "programmer" is a biochemist with little experience in graph theory or traditional programming models and languages. This presents a significant hurdle to overcome when attempting to design or program one of these devices. In this section an intermediate representation (IR) in the javascript object notation (JSON) format is presented along with an abstract and device specific assembly language and a higher level language (HLL). These languages are designed to standardize the interface of the algorithms in this field, and present an easy to use interface for the biochemists. In addition to the application programming, a microfluidic hardware design language (mHDL) based on very high speed integrated circuit hardware design language (VHDL) is presented to enable device and entity designers to describe novel devices at a higher level.

5.2 Assembly Language

The assembly language is lower level language meant to interface more directly with the system. This provides a target for future work in higher level languages, where a newly developed language could compile into this assembly language and be provided as input to the system. This assembly language is divided into two languages:

Abstract assembly language which abstractly describes the functionality that is desired. This assay description is not tied to any specific device architecture or even technology. The operations are described for the functionality that they need to perform, without any underlying knowledge of the architecture or technology. This description needs to go through an entity matching phase during which each operation is matched to an entity type that is needed to perform that operation. If the entity matching phase fails then the description is either malformed, or there do not exist the proper entities in the library to complete that assay.

Device specific assembly language has knowledge of the underlying technology, and potentially even the underlying architecture. This language initially has undergone the entity matching phase, and the required entity types within a given technology are known. At this point the function names and parameters are more precisely defined for the technology that the device will be fabricated in. As the LoC architecture is synthesized, this description is improved, adding in more information about the specific components that will be performing each operation, and including the move operations between those components. Once the device has been synthesized completely, and the assay has been completely scheduled onto it, this description can be used to generate a control program to execute the assay on the fabricated device.

5.3 Biochemical "Programming" Language

In recent years there has been more research being done into formalizing and standardizing a biochemical "programming" language. This formal language would enable biochemists to specify their experiments with more determinism and reduced ambiguity, increasing the reproducibility of experiments. Towards this end several languages were developed: the EXACT description of biomedical protocols [78] attempted to describe the experiment actions that were being taken in something akin to an assembly language, Aqua HLL and Aquacore Instruction Set (AIS) [7, 6] were developed as programming language for the Aquacore software-programmable LoC (SPLoC), a device that is similar to a field programmable gate array (FPGA) but for the LoC domain.

The BioCoder language is a C library developed at Microsoft Research, India, which is intended to standardize the specification and dissemination of biological protocols [11]. A biologist specifies the protocol in C using the BioCoder library; the BioCoder compiler then outputs a step-by-step specification of the protocol in a manner that is fully unambiguous yet the output has the conceptual look and feel of a recipe in a cookbook. This unambiguous recipe can then be disseminated to other biologists in order to enhance clarity and reproducibility within the larger research community. This language seemed to have gained the most traction with numerous assays specified on the open source biology and biological engineering site OpenWetWare.org [1]. The original language was not intended for use with microfluidic, or even lab automation in general, although the original paper does hint at its possible use in those areas. In [62] the BioCoder language was adjusted and targeted towards LoC devices, more specifically to be used as a high level input to this system.

Influenza Detection Assay:

```
The thermocycling protocol applied to the device consists
of 92 degrees C for 30s, and then 35 cycles of the following:
92 degrees C for 5s, 55 degrees C for 10s, and 72 degrees C
for 20s, and finally 72 degrees C for 60s, for a total cycling
time of 22 minutes. A portion of the PCR product (~60nl) is
subsequently subjected to a restriction endonuclease digestion
within the same device. The restriction digest reaction is
performed at 37 degrees C for 10 min.
```

Figure 5.1: English language specification of an influenza detection assay [70].

Figure 5.1 is an English language specification of an influenza detection assay [70]. As can be seen, this is an informal method of specifying an assay, which leads to ambiguities and errors; the BioCoder language addresses these shortcomings.

The language was modified in several ways to target microfluidic LoC devices. First, the syntax was modernized to more closely resemble C++, rather than the older C language. Secondly, portions of the language that were not compatible with LoC devices were eliminated, such as support for solid substances. At present, BioCoder does not support control flow operations, so all loops are explicitly unrolled at compile time. The BioCoder specification of an assay makes no underlying assumption about the target technology (Loc devices, or something else) or about the target architecture (e.g., how many mixers are available, interconnection schemes, etc.). A device-specific compiler that targets known LoC devices must determine if the device has sufficient architectural components and interconnections to execute the protocol.

Figure 5.2 presents the BioCoder specification of the influenza detection assay Fig. 5.1. The fluids are first initialized with their respective names and volumes. Physical resources that perform operations are viewed as "abstract containers," to designate storage of fluid; the synthesis process will allocate physical containers in the LoC device which will store the physical fluid. The remainder of the BioCoder program specifies the operations to be performed;

these operations are compiled into an intermediate representation of the sequencing graph for the described assay. The dependencies ($\mathcal{E}$) are extracted to complete the sequencing graph imposing a partial ordering on the operations.

5.4 Microfluidic Hardware Design Language (mHDL)

The LoC devices are not always derived from a specific application model, at times bio-engineers will want to specify a device or entity model independent of a particular application. These could be novel devices to perform a particular "step" for an assay or to execute an entire class of assays for reuse in a lab setting. To this end, a higher level microfluidic hardware design language (mHDL) was developed in [61].

The mHDL syntax, shown in Fig. 5.3, is based on VHDL and has been designed to be easily extensible to deal with the creation of new entity types as well as the development or discovery of new fabrication technologies. The flexibility of the language comes from the use of a library of entity files during the synthesis process so the mHDL syntax can grow and evolve with the introduction of new LoC entity types and fabrication technologies.

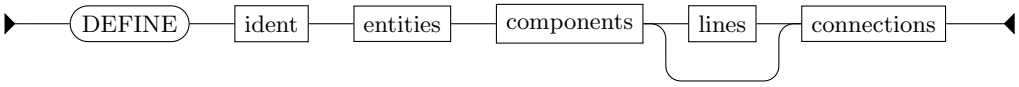
The mHDL file starts by naming the current chip design, shown in the *Design* rule in Fig. 5.3. The *entity types* are then declared in the *entities* section, followed by a list of *components* and the *connections* between those components. The engineer may optionally declare *intermediate lines* to be used, if finer grained control of the routing is desired.

The entities can be either *active* or *passive*, whether they require valves for control or not; an active component is declared *with control* while a passive component is declared *without control*. This annotation simplifies the internal representation and allows the system to identify and work with control and flow layers separately. Entities are declared with the same name as the library file associated with them, but without the file extension. The *components*

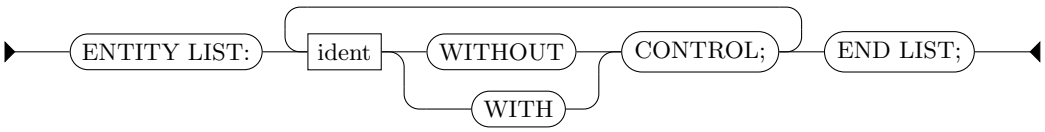
```
static BioCoder* assayFluDiagnosis() {
  BioCoder* FluDiagnosis = new BioCoder("Influenza Detection");
  Container tube1 = FluDiagnosis->new_container(STERILE_MICROFUGE_TUBE2ML);
  Fluid F1 = FluDiagnosis->new_fluid("PCR_Reagents",Volume(240,UL));
  Fluid F2 = FluDiagnosis->new_fluid("DNA",Volume(600,UL));
  Fluid F3 = FluDiagnosis->new_fluid("RDReagents",Volume(500,UL));
  Fluid F4 = FluDiagnosis->new_fluid("ReproGel",Volume(100,UL));

  FluDiagnosis->measure_fluid(F1,Volume(240,UL,tube1);
  FluDiagnosis->measure_fluid(F2,Volume(600,UL,tube1);
  FluDiagnosis->vortex(tube1,Time(1,SECS));

  for(int i = 0;i < 35;i++) {
    FluDiagnosis->incubate(tube1,92,Time(5000,SECS));
    FluDiagnosis->incubate(tube1,55,Time(10000,SECS));
    FluDiagnosis->incubate(tube1,72,Time(20000,SECS));
  }
  FluDiagnosis->incubate(tube1,72,Time(60000,SECS));

  FluDiagnosis->measure_fluid(F3,Volume(500,UL,tube1));
  FluDiagnosis->vortex(tube1,Time(1,SECS));
  FluDiagnosis->store_for(tube1,37,Time(60000,SECS));

  FluDiagnosis->measure_fluid(F4,Volume(100,UL,tube1));
  FluDiagnosis->vortex(tube1,Time(1,SECS));

  FluDiagnosis->electrophoresis(tube1);
  FluDiagnosis->drain(tube1,"B3");
  FluDiagnosis->end_protocol();
}
```

Figure 5.2: BioCoder specification of the influenza detection assay.

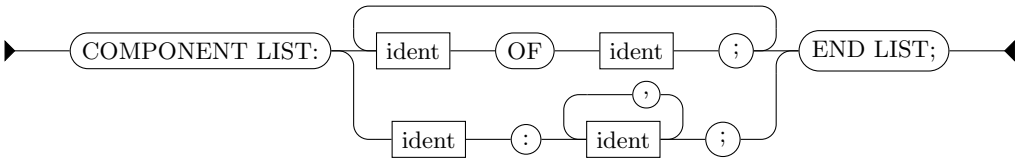
Design:



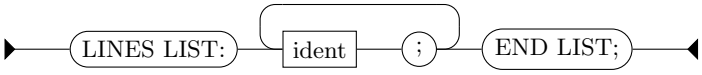
entities:



components:

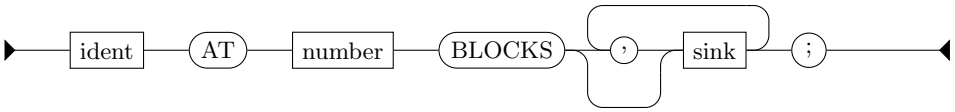


lines:

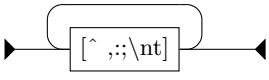


connections:

sink:



ident:



number:

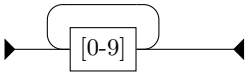


Figure 5.3: A syntax diagram for mHDL. This language provides support for defining the entities, declaring the components, and describing the interconnection between components.

list declares each component and requires an entity type to be specified since the entity type contains much of the information needed by the system. The syntax allows the user to declare one or more components having the same entity type within a single statement.

The *connections* list defines the channels that connect the individual components. Specific intermediate *lines* can be optionally declared for clarity and finer grained control over the routing of the device. Each connection is directional with a single *source* component that injects fluid into it (specified first), followed by a list of *sinks*, which receive fluid. A single source can connect to multiple sinks regardless of the connectivity of the respective components. During the architectural synthesis and switches that are necessary will be inserted. Multiple directional connections with difference sources can be used in lieu of a bi-directional channel.

The *blocks* keyword enables the engineer to provide estimates of the length of a fluid channel to enable simulation of fluid transfer latencies; the exact length is not known until the physical layout of the LoC has been finalized. This construct can also be extended in the future to enforce specific distances between components. Within the fields of biology and chemistry their are specific reactions that must occur within a specified amount of time or the results become nullified. This is an initial step in that direction.

5.5 Technology Files

The system is designed to be able to synthesize LoC devices targeting multiple fabrication technologies. This enables an engineer to compare the trade-offs of devices and decide the best fit for their need. More importantly it allows engineers to develop new fabrication technologies, and to add them to the system with ease. Technology files, which describe the specific requirements and fabrication constraints, are input to the system. To develop a new fabrication technology, a technology file must be created and added to the system. Once the

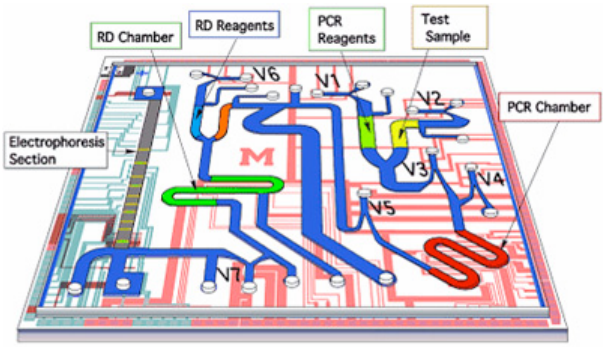


Figure 5.4: Illustration of an LoC that performs influenza detection [70]; the key observation here is that the LoC can be viewed as a netlist of components and their interconnection.

file has been added to the system, that technology can be used on future LoC device designs. Technology files contain the fabrication constraints including spacing requirements, size and I/O restrictions, and timing to open and close a valve. An example of fabrication requirements for the Stanford foundry [3] is shown in Tables 5.1 and 5.2.

5.5.1 Entity Libraries

The different technologies that are made available have different entities which they support and different valves they utilize. The valves are the microfluidic equivalent to gates, and the entities are circuits that are built from these basic gates to perform specific operations, in place of adders and multipliers there exist mixers and heaters. Interestingly enough, the microfluidic equivalent of adders has been developed to perform computation [45]. The entities are designed to perform the operations necessary to execute the assays, and as engineers continue to develop novel ways to perform biochemical operations on microfluidic devices, more entities will need to be inserted into the system. These entities can be either passive, with no valves or controllable elements, or active, with valves and/or controllable elements.

Influenza Detection LoC mHDL Specification

```
define flu_diagnosis:
  entity list:
    storage without control;
    valve with control;
    mixer with control;
    electrophoresis with control;
    PCR with control;
    RDR with control;
    Exhaust with control;
  end list;
  component list:
    valve: V1,V2,V3,V4,V5,V6,V7,A4,A6,B2,RDvalve;
    PCR_chamber of PCR;
    RD_chamber of RDR;
    Electrophoresis_section of electrophoresis;
    B3 of Exhaust;
    Storage: L1,L2,L3,PCR_product,B1B4;
  end list;
  connection list:
    L1 connects to V1 at 10 blocks;
    V1 connects to V3 at 100 blocks;
    L2 connects to V2 at 10 blocks;
    V2 connects to V3 at 120 blocks;
    V3 connects to V4 at 40 blocks;
    V4 connects to PCR_chamber at 40 blocks;
    PCR_chamber connects to V5 at 40 blocks;
    V5 connects to A4 at 110 blocks;
    A4 connects to PCR_product at 10 blocks;
    PCR_product connects to A6 at 10 blocks;
    A6 connects to RDvalve at 50 blocks;
    L3 connects to V6 at 10 blocks;
    V6 connects to RDvalve at 100 blocks;
    RDvalve connects to RD_chamber at 10 blocks;
    RD_chamber connects to V7 at 30 blocks;
    V7 connects to B1 at 10 blocks;
    B1 connects to B2 at 20 blocks;
    B2 connects to Electrophoresis_section at 10 blocks;
    Electrophoresis_section connects to B4 at 10 blocks;
    B2 connects to B3 at 10 blocks;
  end list;
end define;
```

Figure 5.5: The mHDL specification of the netlist from the chip shown in Fig. 5.4

Table 5.1: Valve area and approximate closing pressure from the Stanford foundry [3].

Push-down Valve		
Valve Area	Flow Channel Height	Closing Pressure
100 µm × 100 µm	5 µm	10psi
100 µm × 100 µm	10 µm	10psi
100 µm × 100 µm	15 µm	10psi

Table 5.2: Critical design rules from the Stanford foundry [3].

Parameter	**Value**
Min. overall chip thickness	3 mm
Max. overall chip thickness	7 mm
Min. flow channel height	5 µm
Max. flow channel height	15 mm
Min. possible feature width	15 µm
Max. possible feature height	150 mm
Nominal control channel height	10 µm or 25 µm
Min. spacing between borders	2 mm
Min. spacing between punch holes (20 gauge)	1500 µm

An entity $E = (F, H)$ is defined with a list of the biochemical operations, or functions, it can perform F and the dimensions of the bounding box of the footprint of the entity $H = w \times h \times d$. The locations of the Input/Output ports along the edge of the bounding box must also be defined for the routing step. If the entity is an active entity (contains valves), the location of those valves must also be specified. These locations are used when routing the control layer of the device. The functions also contain a list of actuation sequences that define the order in which the valves must be actuated to perform that specific operation. These functions are used by the compiler when mapping a given application model to an architecture model, or when synthesizing a new device design from an application model.

Part III

Flow Layer

Chapter 6

Flow Layer Synthesis

An LoC device is made up of two logical types of layers, flow and control. These layers are similar in many ways, but also provide their own unique opportunities and constraint. In this chapter the flow layer is discussed in detail, and the algorithms for the placement and routing of these layers are introduced. These algorithms may sometimes be reused during the control layer synthesis, discussed in the relevant chapter.

A device netlist D, represented as a graph, can be partitioned into subgraphs, $D_i \subset D$, to represent the different layers, where D_i are disjoint sets such that $\Cup_{D_i} = D$. The notation D_{f_i} is used to represent the flow layer(s) containing a subset of the switches $s_i \in S$ and components $m_i \in M$. The flow layer contains all of the switches and the components of the device, the associated control layer(s) will be synthesized from the resulting placed and routed flow layer.

The flow layer netlist is viewed abstractly as a graph, and so can be thought of, to some extent, as analogous to a circuit in VLSI. This abstraction allows us to borrow, and adapt, algorithms from the well established EDA industry for the placement and the routing of the device. Microfluidic LoC devices have been designed directly using electric circuit analogies by Kwang W Oh et. al [69] and traditional VLSI simulation software has been used to verify

industry designs. However, unlike traditional VLSI, the routing latencies of fluids makes up a non-trivial portion of the assay execution time. Additionally, once a channel has been used to route a fluid, that channel may be contaminated and unable to be used to route future fluids until it has been washed. These constraints complicate the placement of routing of these devices.

The following sections first introduce several placement algorithms, simulated annealing and planar embedding. The routing algorithms, maze routers and network flow are then introduced. Finally, a combined scheduler and router is introduced and integrated washing of the device is discussed.

6.1 Simulated Annealing

The process of design space exploration involves altering a design to improve the quality of the design. In greedy algorithms this process of continually improving the design can lead to finding locally optimal designs, shown in Fig. 6.1. At this point the algorithm, unable to accept a worse design, is stuck with a suboptimal design. The simulated annealing process uses randomization to overcome this shortcoming. The components on the device are manipulated randomly using prescribed 'move" operations. The cost of the resulting solution is then calculated, and if the solution is better it is immediately accepted. If the new solution is worse, however, it may still be accepted based on a random number. This allows the design space exploration to "climb" out of locally optimal valleys and continue searching for a more globally optimal solution. As the name suggests, the algorithm follows a predefined "cooling" schedule, where it will initially accept worse solutions with a high probability, but as the algorithm progresses a worse solution is less likely to be accepted, allowing the solution to progress towards a more globally optimal solution.

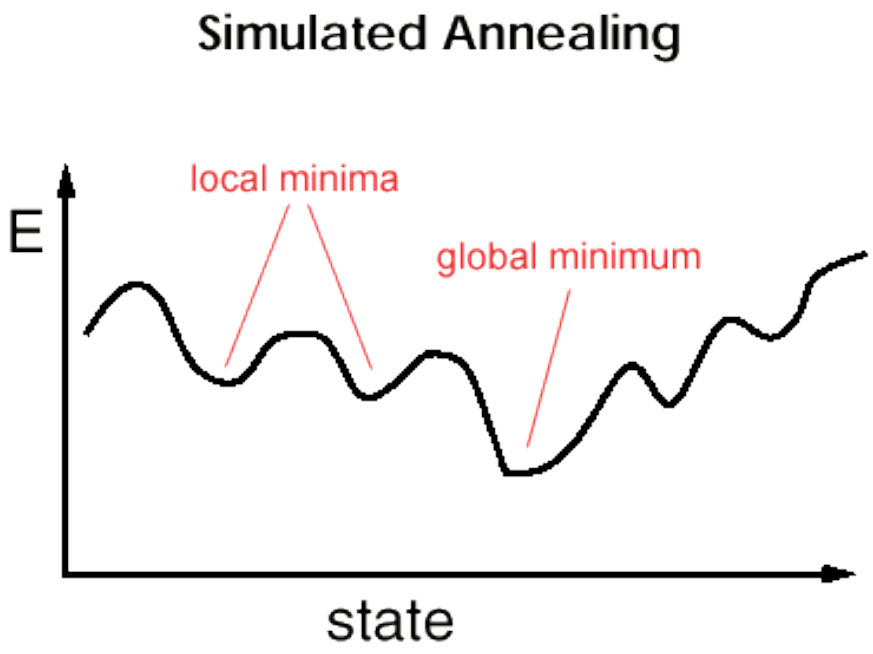


Figure 6.1: A design space typically contains many locally optimal solutions. The process of simulated annealing allows for worse solutions to be selected to "climb" out of these valleys and find the globally optimal solution.

Simulated annealing is a meta-heuristic which describes a process more so than an actual algorithm. To implement simulated annealing several things need to be defined 1. initial solution, 2. the "move" operations, 3. a cost function to determine solution quality, and, 4. a cooling schedule Due to it's flexibility, simulated annealing has had a long history of success in a number of fields including number partitioning [47], graph coloring [46], and more importantly in standard cell VLSI [76] and FPGA placement [85]. Prior work in microfluidics also claims to use simulated annealing [68], although the details are not discussed. This chapter discusses the details of a simulated annealing implementation for placement of the flow layer of LoC devices.

6.1.1 I/O Region Segregation

The physical area of the LoC device is represented as a grid, where the pitch defines the granularity of the perturbation operations; in this chapter several pitches are utilized and discussed. The components m_i occupy multiple nodes in the grid. As shown in Fig. 6.2a, the grid is partitioned into four region types: input, output, center, and invalid. The placer is constrained to place I/O punch holes in the input and output regions respectively, and all other components in the center. The invalid regions are isolated in the corner of the chip and no components or I/Os may be placed there as placing components in the chip corners often leads to routing failures, as shown in Fig. 6.2b. It is assumed initially that all components occupy one rectangular unit of grid area; this constraint is relaxed in Section 6.1.4. Non-rectangular components, such as the circular mixer in Fig. 4.3, are represented by their bounding-box as defined in the entity library. The channel routes are not known during placement, so the routing estimate assumes all channels start at the center point of the source component, and end at the center point of the sink component, as shown in Fig. 6.3a. This provides an estimate on the

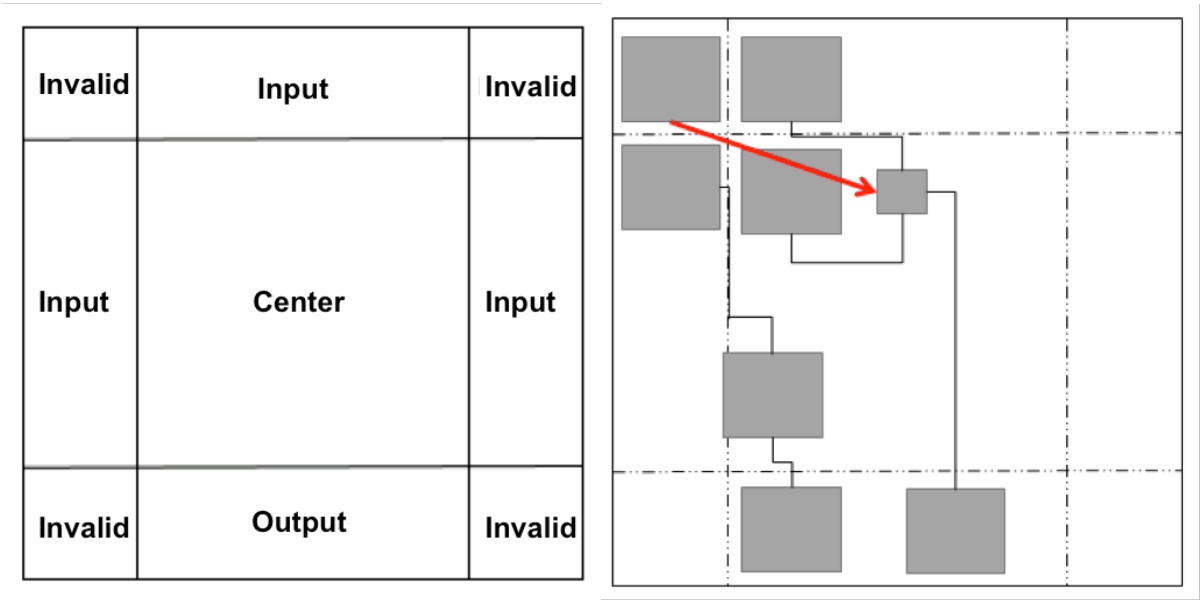
length of the channel, using the manhattan distance, and the number of intersecting channels. Figure 6.3b shows the resulting LoC layout after routing has been performed.

6.1.2 Perturbation

This simulated annealing placer implementation employs two different perturbation operations, shift and move. First, a component $m_i \in M$ is chosen at random. Then a shift or move perturbation is chosen randomly and with equal probabilities. Throughout the simulated annealing process, all solutions must be legal, therefore any perturbation that shifts or moves m_i into an incompatible grid region or off the grid, is suppressed. The shift direction (up, down, left, right) is chosen randomly with equal probabilities. Similarly, if a move operation is selected, the target location in the grid is chosen randomly; all target locations have equal probabilities. If the target position randomly chosen for m_i is empty, then m_i is moved there; if the target position contains another component, m_j, then m_i and m_j swap positions. This becomes more complex when heterogeneous component sizes are considered in Section 6.1.4. After each perturbation, the cost function that characterizes the placement solution is updated, and the perturbation is accepted or rejected: if the cost function improves, then it is accepted outright; if not, it may still be accepted based on a probabilistic calculation, as per the typical behavior of simulated annealing [76].

6.1.3 Cost Function

The cost function assesses the quality of the placement of the components on the grid; the foremost concern is route-ability. Three metrics are considered: (i) the estimated number of intersections, (ii) total edge length, and (iii) the sum of the squares of each edge length.

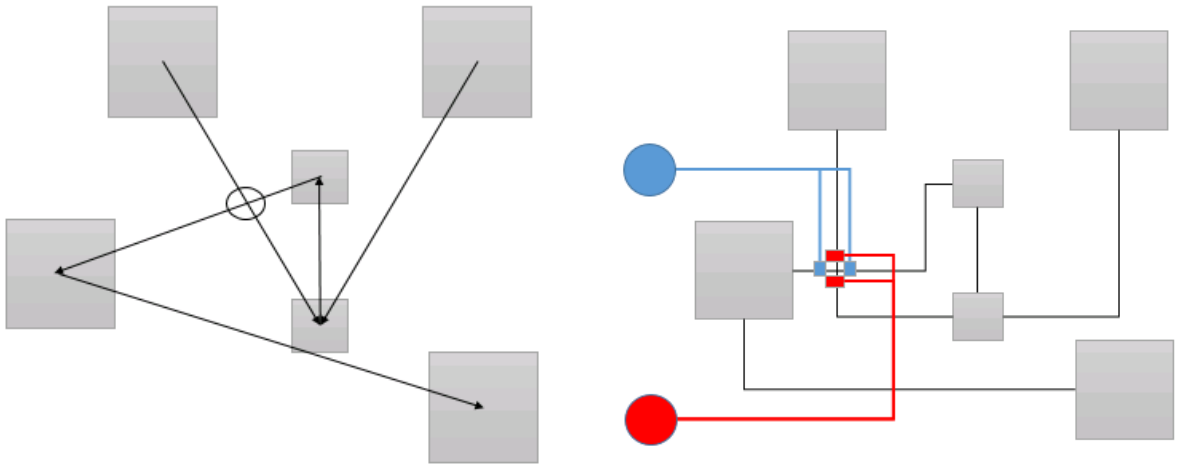


(a) (b)

Figure 6.2: (a) The mLSI grid is partitioned into input, output, center and invalid regions. (b) Placing a component in an invalid region may cause a routing failure.

The intuition behind reducing the number of intersections as each intersection introduces additional valves, and potentially additional external pressure sources, as shown in Fig. 6.3b. In most fabrication technologies, there is a fixed limit on the number of punch holes that a device can support, defined by the foundry design rules. At the Stanford foundry [3], this limit was at thirty-five at the time of writing, which can impose a significant constraint in fabrication. Additionally, each additional I/O adds another net to the control layer, which complicates the control layer synthesis.

The intuition underlying the total edge length metric is two-fold. First, shorter fluid channels allows for lower fluid transportation times, which represent a non-trivial portion of the assay execution time. Second, if a significant proportion of the device area is dedicated to fluid channels, the structural reliability of the flow layer becomes an issue. A channel is essentially



(a) (b)

Figure 6.3: (a) Fluid channels are represented as direct lines between the centers of components. The circle shows an intersection point between two lines. (b) After routing, the intersection is converted to a 4-microvalve switch, requiring two external pressure sources (circles) and two control channels.

the absence of PDMS and if too little PDMS is present there is a higher likelihood that the flow layer will collapse on itself becoming unusable. Reducing the total channel length thereby increases the amount of PDMS present, which increases the likelihood of successful fabrication.

The intuition of summing the squares of the length of each edge was to penalize very long routes, thereby reducing the variance in route length, leading to fairly equal spacing among components. This could potentially improve the route-ability of the device.

Equations

Let $D = (V, C)$ be a graph representing the netlist of the LoC device being synthesized. Let $c = (p_i, p_j)$ be an edge connecting the center points p_i and p_j of two components m_i and m_j, where each point $p_k = (x_k, y_k)$ is defined in terms of its x- and y- coordinates. Let N be the number of intersections in D. The intersection point between two non-parallel infinite lines defined by segments $c_1 = (p_1, p_2)$ and $c_2 = (p_3, p_4)$, respectively, is the point $p^* = (x^*, y^*)$, where

$$x^* = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}, \text{ and } y^* = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}} \tag{6.1}$$

The length of an edge segment $c = (p_1, p_2)$ is assumed to be the Manhattan Distance (MD) between the two points:

$$MD(c) = |x_1 - x_2| + |y_1 - y_2| \tag{6.2}$$

The total edge length L and total squared edge length S are

$$L = \Sigma_{c \in C} MD(c), \text{ and } S = \Sigma_{c \in C} MD(c)^2 \tag{6.3}$$

The objective function, $F(D)$, is computed as follows:

$$F(D) = \alpha N + \beta L + \gamma S, \tag{6.4}$$

where α, β, and γ are user-specified weights, which can be modified to alter the relative importance of each metric.

Implementation Details

The time complexity of computing the number of edge intersections is $O(n^2)$ in the general case; however recalculating the cost for each edge after each move is unnecessary. A two-part cost function is implemented to speed up the annealing process. First, each edge maintains a list of the other edges that it intersects. When a component is moved or swapped, any change in the set of intersecting edges will exclusively involve edges incident on the components that move. $F(D)$ is computed explicitly after the initial placement; subsequently, it is incrementally updated.

Let C be the subset of edges that are incident on the component(s) that are moved during a perturbation. Let N denote the number of edge intersections involving edges in C,

$$L' = \Sigma_{c \in C'} MD(c), \text{ and } S' = \Sigma_{c \in C} MD(c)^2 \tag{6.5}$$

After moving all of the component(s) involved in the perturbation, let N'', L'' and S'' respectively denote the updated values of N, L and S that are recomputed after moving the component(s). Then the updated metrics after the perturbation are:

$$N_{new} = N - N' + N'', \tag{6.6}$$

$$L_{new} = L - L' + L'', \text{ and} \tag{6.7}$$

$$S_{new} = S - S' + S'' \tag{6.8}$$

yielding an updated value for $F(D)$, denoted as $F_new(D)$. If $F_new(D)$ is an improvement (i.e, if $F_new(D) < F(D)$), then the perturbation is accepted; if not, it may still be accepted probabilistically in accordance with the operating procedure of simulated annealing.

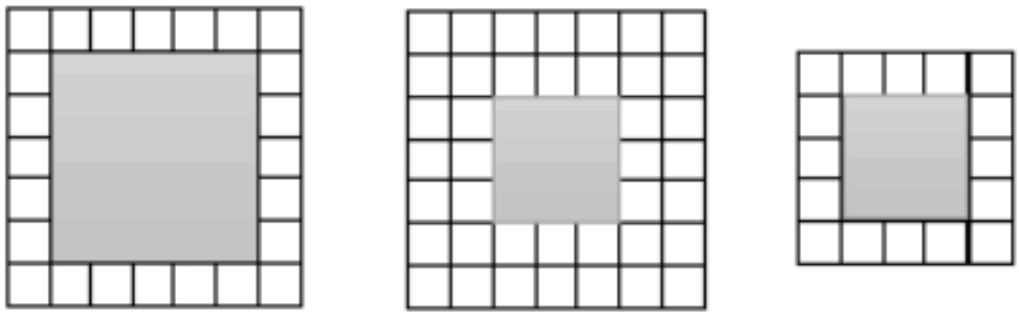
6.1.4 Heterogeneous Component Geometries

Initially it was assumed that all components have uniform size; to achieve this uniformity, the placer preemptively pads smaller components with extra cells to ensure that all components are of the same size, as shown in Figs. 6.4a and 6.4b. This simplified the perturbation process, but yields poor area usage for LoC architectures with a large number of relatively small components. In some cases, the placer could not find legal solutions for small LoC devices easily placed by hand using exact component sizes.

To introduce heterogeneity, the grid pitch is reduced, allowing finer grained control over the location and size of components. The components are still assumed to be rectangular, using the bounding box defined in the entity library. The grid pitch is now defined as the greatest common divisor (GCD) of the component widths and heights and the minimum spacing width between components (a foundry design rule). Component dimensions are now recomputed as integer multiples of the grid pitch. The heterogeneity in component sizes introduces several complications during the perturbation operations, specifically the "swap" step within the move perturbation operation.

Heterogeneous Swaps

Let a *block* be a node in the grid. The annealer randomly selects a component m_i and a target block b, which must be outside of the region encompassing m_i. If b is contained within a component, m_j, then the annealer sets b to be the top-left corner of m_j. Next,



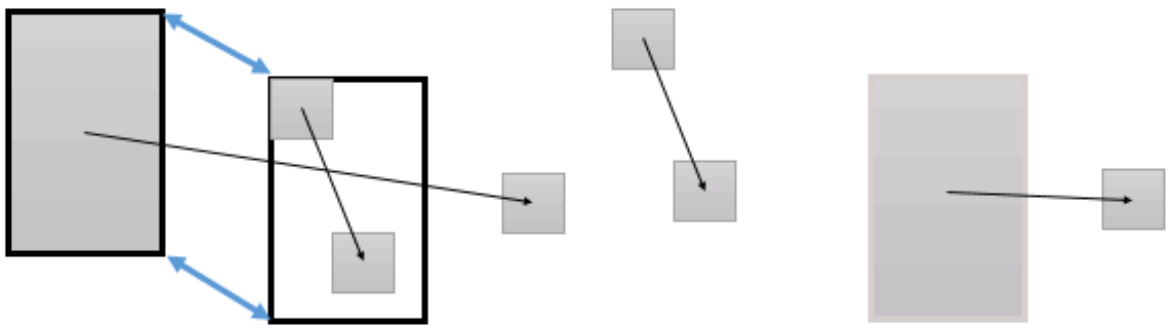
(a) (b) (c)

Figure 6.4: (a) A large component; (b) if all components require a uniform size, smaller components are padded with extra cells; (c) alternatively, the same amount of padding is used for each component, yielding heterogeneous sizes.

the annealer creates a region R with b as the top-left corner, and dimensions equal to the maximum dimensions of m_i and m_j. If R contains blocks (nodes) from multiple grid regions (e.g., input/center) then the swap aborts preemptively. If all components with at least one block in R are fully contained in R, then the swap is legal; all components within R are swapped with m_i, as shown in Fig. 6.5. If any component with at least one node in R is not fully contained within R, then the swap is aborted, as shown in Fig. 6.6.

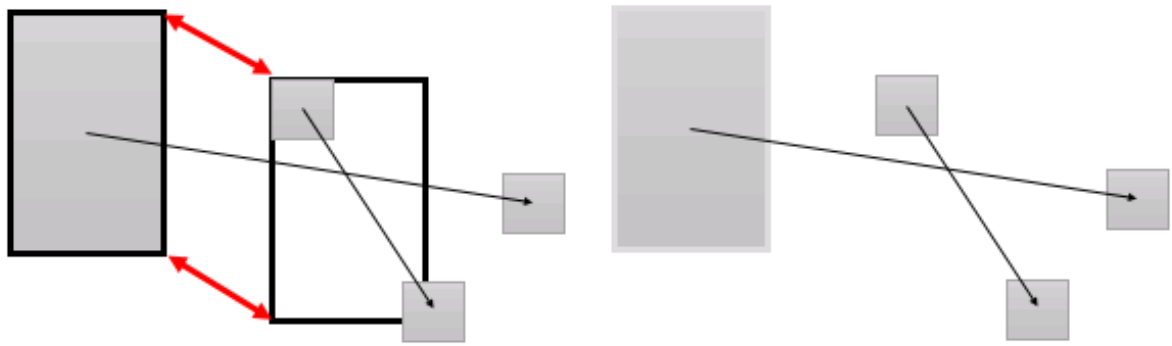
Shifts

The annealer randomly selects a component m_i and a direction d (up, down, left, right). The shift tries to move m_i one block in the direction d. A region R is created starting in direction d from the top-left of m_i. If R contains no components other than m_i, the shift is legal (Fig. 6.7); otherwise it is aborted (Fig. 6.8).



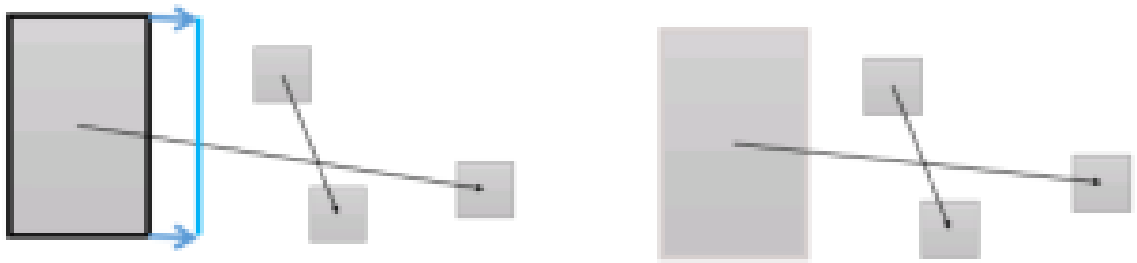
(a) (b)

Figure 6.5: Illustration of the region formation process for a successful swap operation (a); since both components are contained within the region, the swap proceeds (b).



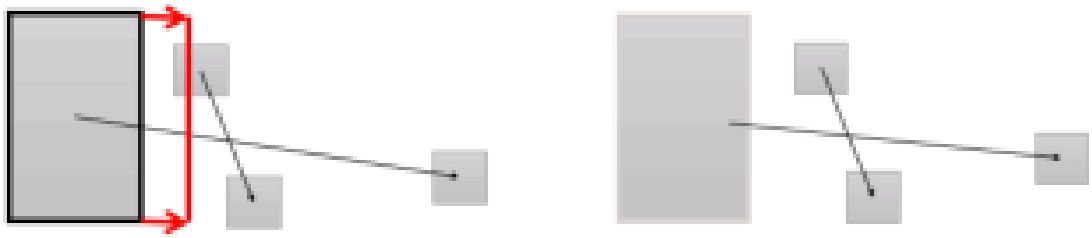
(a) (b)

Figure 6.6: Illustration of the region formation process for a failed swap operation (a); since one component is partially contained within the region, then swap aborts (b).



(a) (b)

Figure 6.7: Illustration of the region formation process for a successful shift operation (a); since there are no components in the target region, the shift proceeds (b).



(a) (b)

Figure 6.8: Illustration of the region formation process for a failed shift operation (a); since there is a component in the target region, the shift aborts (b).

6.1.5 Component Spacing

To ensure that components have a valid placement, there must be space between them to accommodate fluid channels for routing. Unlike standard cells, it is not possible to dedicate different layers of metal (PDMS) for logic (components) and wires (fluid routing channels). The minimum spacing requirement to ensure successful fabrication is a design rule provided by the foundry. Spacing beyond the minimum may be necessary to achieve route-ability.

Let Δ be the spacing constraint, $m_i.h$ and $m_i.w$ be the height and width of the component, $B.h$ and $B.w$ be the height and width of a block in the grid. Then the number of blocks required for the height (H) and width (W) of a component, including spacing, is:

$$H = \left\lceil \frac{m_i.h + \Delta}{B.h} \right\rceil, \text{ and } W = \left\lceil \frac{m_i.w + \Delta}{B.w} \right\rceil \tag{6.9}$$

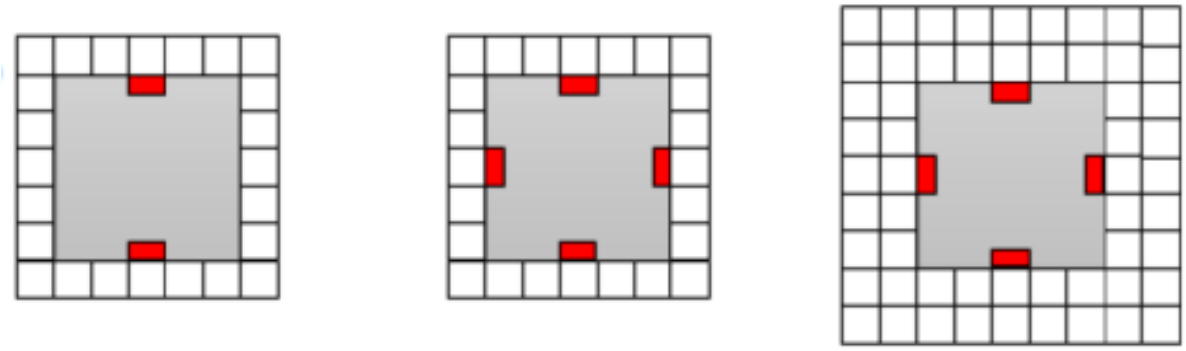
The spacing rules from Eq. (6.9) work fairly well, but led to routing failures when the LoC device netlist incorporated components that connect to a large number of fluid channels. The components with higher connectivity must have more channels route in close proximity, while components with lower connectivity typically have fewer channels in close proximity.

Let $f(m_i)$ be the number of fluid channels connected to component m_i. To account for $f(m_i)$, the components height (H_f) and width (W_f) are

$$H_f = \left\lceil \frac{m_i.h + \Delta f(m_i)}{B.h} \right\rceil, \text{ and } W_f = \left\lceil \frac{m_i.w + \Delta f(m_i)}{B.w} \right\rceil \tag{6.10}$$

This provides sufficient spacing to route fluid channels around components with a high degree of fluidic connectivity, while simultaneously reducing excess padding around components with lower connectivity.

As shown in Figs. 6.9a and 6.9b, two components of equal size have equal of padding, regardless of the number of I/O ports (red). In Fig. 6.9c, a component with four I/O ports, instead of two, gets an extra layer of padding to improve route-ability.



(a) (b) (c)

Figure 6.9: Components receive equal padding, regardless of the number of I/O ports (a)/(b); extra padding added to a component with four ports to improve route-ability (c).

6.1.6 Initial Placement

The initial placement of components may affect the ability of the simulated annealing-based placer to find a good solution. Here, two greedy heuristics to select the initial placement are considered.

Random Initial Placement

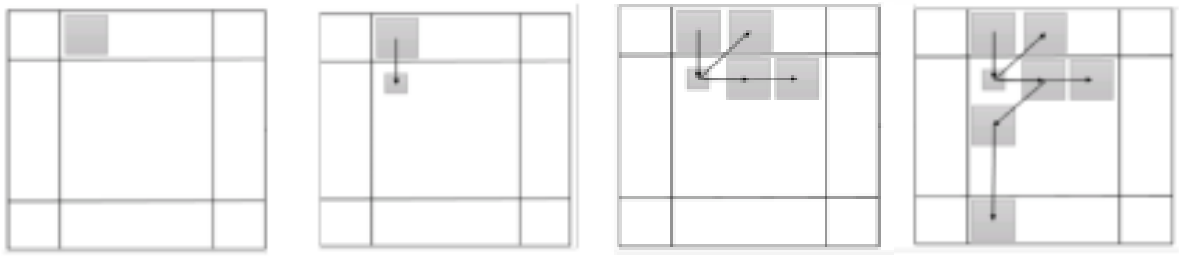
The components are processed from largest to smallest to limit the likelihood of failure due to fragmentation of the grid area if small components are placed before large components. For each component, the placer determines if a valid position is available; next it generates a random location in an appropriate region (input, output, center) for the component: if the location is legal, then the component is placed there; if not, a new random location is generated and the process repeats until a legal location is found. A failure is declared if there are no valid positions detected for a component.

Directed Initial Placement

The first input component is placed in the top-left position in the input grid region. The algorithm then iterates over the list of outgoing edges, and each component is placed nearby in the appropriate grid region (input, output, center). It maintains three variables that represent the starting integer position of the input, output, and center regions in the grid. When a component is selected, the variable for its compatible region selects the next available location. The components dimensions determine the number of grid blocks in the horizontal and vertical directions to search. The search continues until a valid placement or the end of the region is found; placement fails in the latter case. This process repeats until all components are placed, as shown in Fig. 6.10.

6.1.7 Experimental Results

Although many LoC devices have been published in bioengineering literature, netlist representations are unavailable for most of them. A set of netlists made publicly available by researchers at the Technical University of Denmark [2]. Each netlist is placed using simulated annealing and routed using a variant of Hadlocks algorithm for mLSI technology, as suggested by Minhass et al. [68].



(a) (b) (c) (d)

Figure 6.10: The first four steps of a directed initial placement

Standard parameter values for simulated annealing implementation were used: an initial temperature of 100°, with a cooling rate of 5%, ten thousand successful moves at each temperature level, and a freezing temperature of 1°. These parameters were used without modification for all experiments, and the results of the implementation decisions are reported using the following metrics:

Success Rate: Each netlist is placed ten times with ten different random number seeds. The success rate is the number of times Hadlocks algorithm successfully routes the placed chip.

Number of Intersections: The number of intersections estimated by the placer, e.g., Fig. 6.3a, and the actual number of intersections occurring after routing, e.g., Fig. 6.3b are reported. Results are averaged across all (successful) runs. Hadlocks algorithm is ineffective at avoiding intersections and a more intelligent router may be able to provide more success cases with fewer intersections. Hadlocks algorithm was chosen to be consistent with prior work on LoC physical design [68].

Total Channel Length: The sum of the lengths of all fluid channels after placement and routing, i.e., term L in Eq. (6.3).

Random vs. Directed Initial Placement

The first set of experiments (Fig. 6.11(a)/(b)) vary the weights of the metrics in the objective function using random and directed initial placement. Two trends are observed: (1) random initial placement tends to yield higher success rates than directed initial placement; and (2) equally weighting the number of intersections and total edge length generally yielded the best success rates, especially when combined with random initial placement.

The rest of the experiments exclusively use random initial placement and objective function $F(D) = 500N + 500L$.

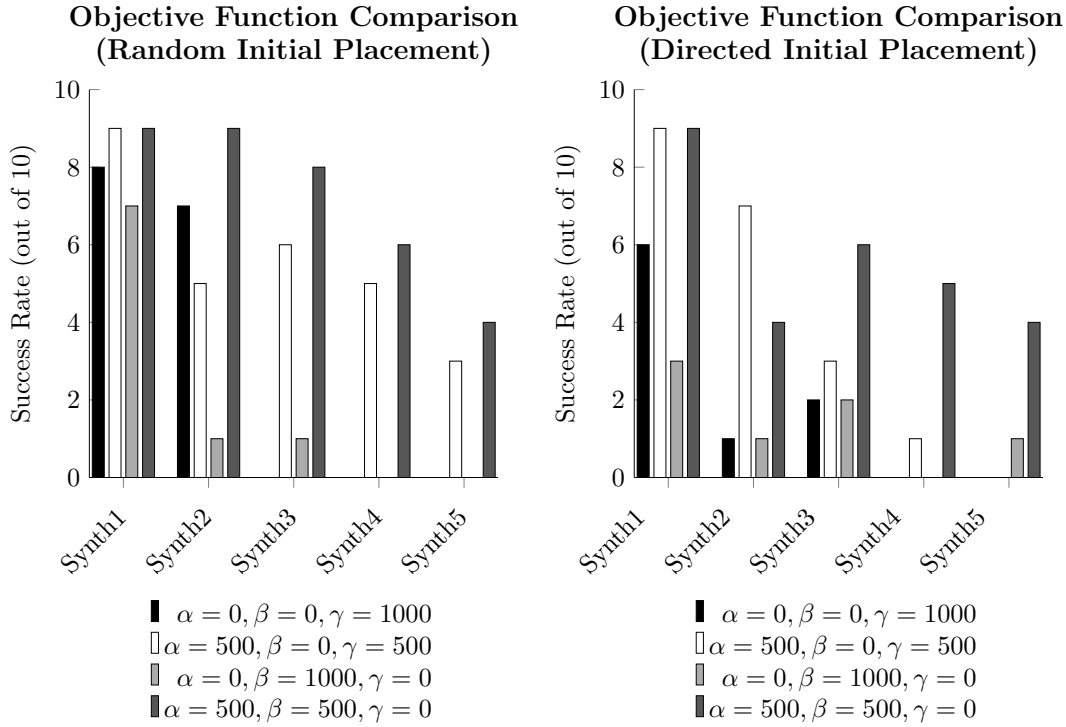


Figure 6.11: Comparison of different objective functions using random (a) and directed (b) initial placement; recall from Eq. (6.3) that the objective function is $F(D) = \alpha N + \beta L + \gamma S$, where N is the number of intersections, L is the sum of the edge lengths of all fluid channels, and S is the sum of the squares of the edge lengths of all fluid channels.

Impact of I/O Padding on Components

Figure 6.12 reports the impact of scaling the I/O padding around components in accordance with Eq. (6.10). Using I/O padding positively affects the success rate, suggesting that it should be used primarily for netlists that are otherwise challenging to route. Routing with Hadlocks algorithm increases the number of intersections compared to the placers estimate. For Synthetic 1-3, which were easier to route, I/O padding increased the number of intersections, while marginally increasing total routing channel length; however, for Synthetic 4-5, which were harder to route, I/O padding reduced the number of intersections after routing, while significantly shortening total routing channel length.

Impact of I/O Region Segregation

Figure 6.13 reports the impact of I/O region segregation (Section 6.1.1) on the success rate and number of fluid channel intersections. I/O region segregation increases the success rate and reduces the number of intersections (post-routing) for all benchmarks. In one case (Synthetic 5), all routing attempts without segregation failed. This experiment was performed with I/O padding.

Summary

To summarize, the best configuration of the simulated annealing placer uses: (1) random initial placement; (2) objective function $F(D) = 500N + 500L$; (3) I/O padding; and (4) I/O region segregation. The runtime of the simulated annealing algorithm ranged from 2.3 minutes (Synthetic 1) to 18.6 minutes (Synthetic 5). These reported runtimes are not absolute, and depend on the temperature schedule, as well as other simulated annealing parameters.

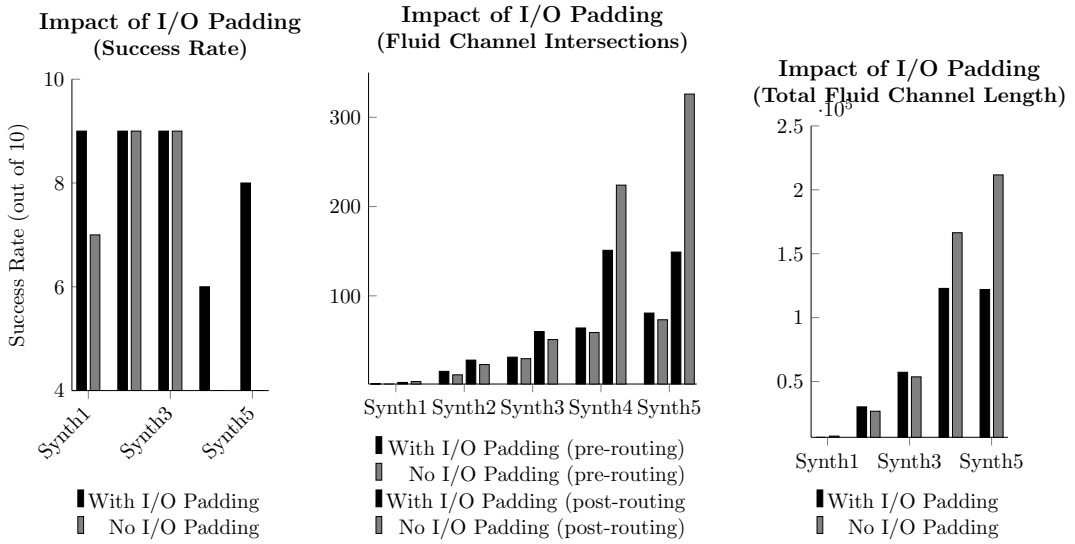


Figure 6.12: The impact of I/O padding on (a) success rate, (b) the number of intersections, and (c) total fluid channel length.

6.2 Planar Placement

Planar Graphs

A graph D is planar if it can be *embedded* in the plane in such a manner that edges only cross at their endpoints. In the context of LoC devices, this means that the netlist can be placed and routed such that fluid routing channels intersect only at components (represented, for now, as points). Every planar graph also admits a *straight line planar embedding* in which the planar graph property is preserved and all edges can be drawn as straight line segments.

Algorithmic planarity testing is typically based on an alternate, but equivalent definition of planarity: graph D is planar if and only if it does not contain the specific graphs K_5 or $K_{3,3}$, Fig. 6.14, as minors, where a minor is a graph H that can be obtained from D by deleting vertices and/or deleting or contracting edges [53].

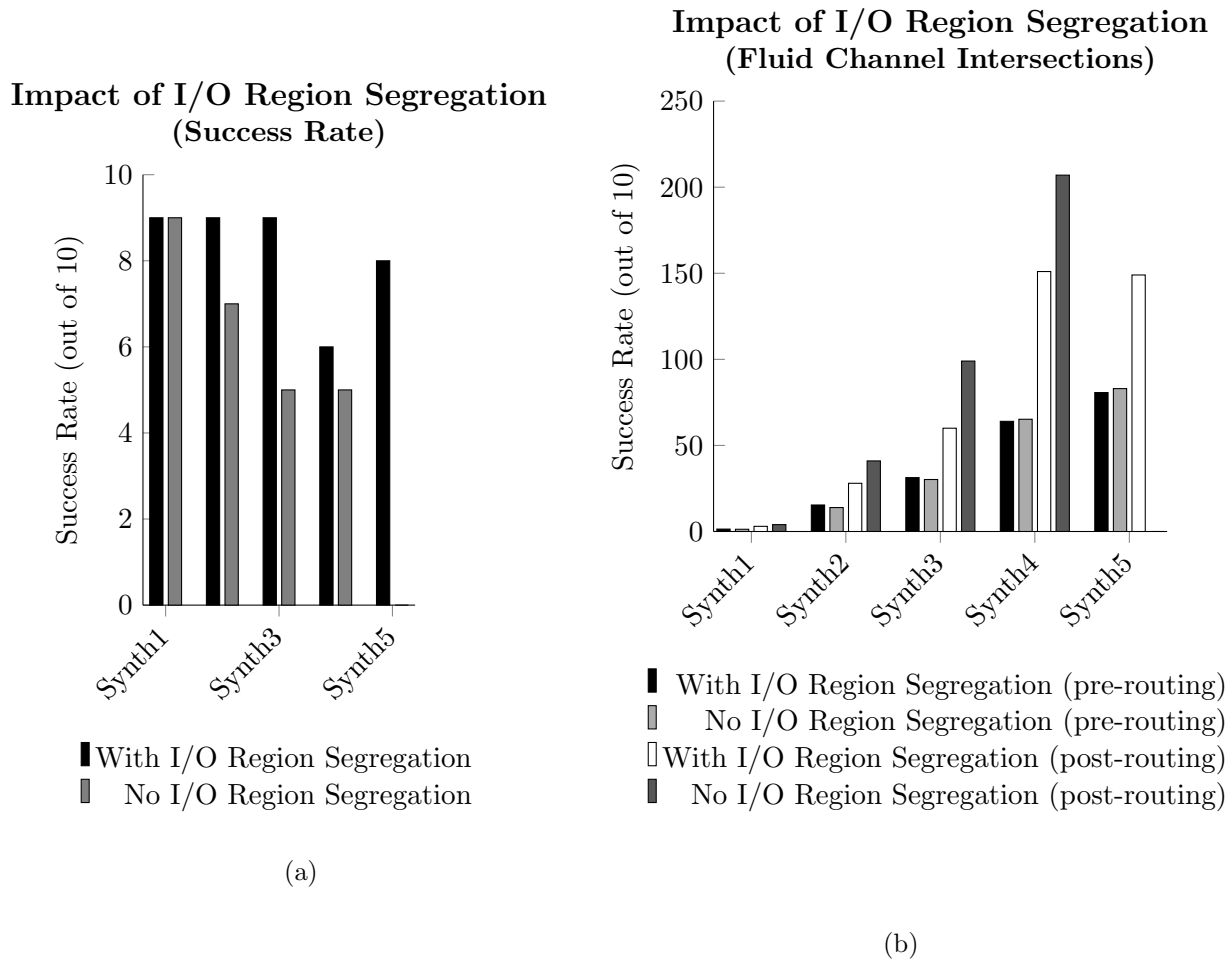


Figure 6.13: Success rate (a) and intersection count (b) for LoC placement with I/O region segregation. Synthetic 5 benchmark failed to route without I/O region segregation.

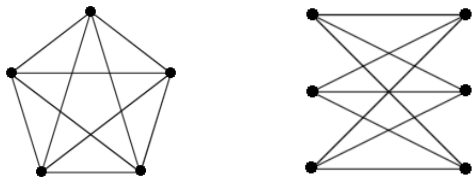


Figure 6.14: A graph with K_5 (a) or $K_{3,3}$ (b) as a minor cannot be planar.

Implications for LoC Device Technology

A legal placement and routing solution for the flow layer of an LoC device is essentially a planar embedding that treats vertices as components with dimensions and area, rather than points. In this context, a legal planar embedding means that components do not overlap one another, routed fluid channels do not intersect components, and routed fluid channels do not intersect one another. Our approach is to take a planar graph embedding and convert it into a planar LoC device embedding, as described in the next section.

6.2.1 Planar Embedding Algorithm

Straight Line Planar Embedding

The process starts with a graph $G = (V, C)$ representing the netlist of components V and their connections C; vertices do not yet have dimensions or area. The first step is to make D fully connected, and check for planarity using the Boyer-Myrvold method [16]. If D is planar, then it is transformed to be biconnected and maximally planar. The vertices $v_i \in V$ are then ordered canonically and the Chrobak–Payne straight line embedding algorithm [22] is invoked to obtain a straight line planar embedding. These steps were implemented using the Boost Library; Fig. 6.15 provides a high-level overview.

Component Expansion

The straight line embedding does not account for the size or dimensions of components. To create a valid LoC device embedding, a two pass method is applied to expand components and remove any overlap between them. The first pass sorts the components $m_i \in M$ by their x_i coordinate in ascending order, and expands each component by its width w_i. All subsequent components $m_j \in M$, where $j > i$, are shifted in the positive x direction by w_i; $x_j = x_j + w_i$. The second pass of the expansion applies the same steps along on the y-axis, while expanding and shifting components based on their heights, rather than their widths. This guarantees that components do not overlap after expansion, as shown in Fig. 6.16.

In practice, the component expansion step rarely preserves the straight line planar embedding that was computed previously. Figure 6.16 shows how the component expansion invalidates the straight line planar embedding. Moreover, there is no direct mechanism to assign fluid channels to component ports after expansion. A routing algorithm is still required to complete the layout of the LoC device.

6.2.2 Results

The Planar placement (PP) algorithm is compared to an Incremental Cluster Expansion (ICE) [82] and the Simulated Annealing (SA) algorithm from the previous section. Two separate routers are used to obtain results, Hadlock's maze routing algorithm (HR) [68] and a Network Flow based routing algorithm (NFR). Every combination combination of placer and router is compared to determine which combination works best. The results are based on the outcome of the router since the placement will directly effect how well the router performs. A comparison is then made between the number of added intersections since adding additional intersections will generally invalidate the design.

The Planar placer (PP) will place points initially, and then expand afterwards, which determines the device size automatically. The Incremental Cluster Expansion (ICE) method will place around a center point and expand from there, which will also determine the device size. *Note: A maximum size restriction was not placed on the device size for these placers though it would be trivial to implement.* The Simulated Annealing (SA) based placer, however, requires the device size as input. The tests were initially run on the planar placer, and then the device sizes from the resulting placement are utilized for the simulated annealing placer.

The algorithm combinations are run on two sets of benchmarks. The first set is a synthetic set of benchmarks first published by [68]. These benchmarks are originally non-planar, however non-planar netlists are currently impossible to fabricate. To make these synthetic benchmarks planar we manually add switches to the netlist to make it artificially planar. The second set of benchmarks is based on devices that have been previously designed by hand and fabricated in [56, 84, 73]. The AquaFlex-3b and AquaFlex-5a benchmarks were derived from images of the devices developed by Microfluidic Innovations LLC. The actual device architectures are proprietary intellectual property of Microfluidic Innovations LLC. and are therefore not cited in this paper.

The Network Flow based routing algorithm (NFR) does not allow for intersections to be added and so results of the Incremental Cluster Expansion placer with the Network Flow based router (ICE+NFR) or the Simulated Annealing placer with the Network Flow based router (SA+NFR) are not reported as those algorithms typically introduce intersections, causing a routing failure. The Hadlock's based router (HR) is able to allow additional intersections, perhaps too liberally; allowing it to route on the non-planar placements generated by ICE and SA. It also caused intersections when routing on the planar placement that is generated.

As shown in Fig. 6.18, the Planar placement method combined with the Network Flow based router (PP+NFR) is the only method capable of creating a placement and routing that is valid for fabrication. While the Planar placement combined with the Hadlock's based router (PP+HR) reduced the number of intersections by 26% versus ICE+HR on average and 40% versus SA+HR on average, even this does not reduce intersections enough to allow for fabrication. The Urbanski et al. [84] test case adds the fewest number of intersections using the PP+HR method, adding only 17. This would add 34 control lines to support these new switches, which with the minimum two input valves that are needed in the flow layer to make a useful device violates Stanford's thirty-five external pressure source maximum for fabrication [3].

Planar embedding increased the total device area by approximately one order of magnitude in comparison with the other approaches (detailed results are omitted due to space limitations). The reason for the large area is the shifting that occurs during the component expansion process. The indiscriminate shifting of all components on the device introduce unnecessary space on the device. Despite the area overhead, it is important to recognize that the proposed approach is the *only* technique published, to date, that yields planar layouts that can be fabricated; thus, there is absolutely no benefit to approaches that reduce area while sacrificing legality.

6.2.3 Conclusion

At present, LoC device fabrication technology only allows for the creation of LoCs with one flow layer. Existing algorithms for LoC physical design ignore this constraint; although they produce designs with small areas and short wirelength, the layouts are non-planar, and therefore require multiple flow layers or too many introduced valves, which exceeds the present

capabilities of modern LoC device fabrication. This algorithm can produce planar LoC device layouts that adhere to the constraint of a single flow layer;

Require: $D := (V, C)$ an undirected graph

Ensure: $D := (V, C)$ with each $v_i \in V$ placed

1: $D := make\_connected(D)$

2: **if** $!boyer\_myrvold\_planarity\_test(D)$ **then**

3: $\quad exit()$

4: $D := make\_biconnected\_planar(D)$

5: $D := make\_maximal\_planar(D)$

6: $X := planar\_canonical\_ordering(D)$

7: $D := chrobak\_payne\_straight\_line(D, X)$

Figure 6.15: Chrobak Payne straight line embedding from the Boost library. The function calls shown here are Boost library calls [4].

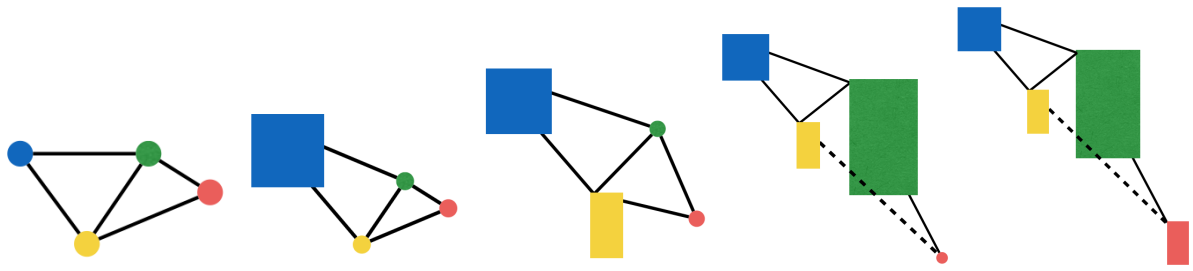


Figure 6.16: (a) The original graph, (b-e) expands the components one at a time to their full size. The dotted line represents the straight line connection that has been invalidated because of the expansion.

Benchmark	Components	Connections
Synthetic1	21	21
Synthetic2	15	21
Synthetic3	34	33
Synthetic4	34	34
Synthetic5	46	45
Synthetic6	62	64
AquaFlex-3b	15	14
AquaFlex-5b	17	16
Li et al. [56]	13	12
Urbanski et al. [84]	13	12
Rhee and Burns [73]	30	37

Figure 6.17: The number of components and connections for each benchmark, Synthetic on top followed by Real Life on the bottom

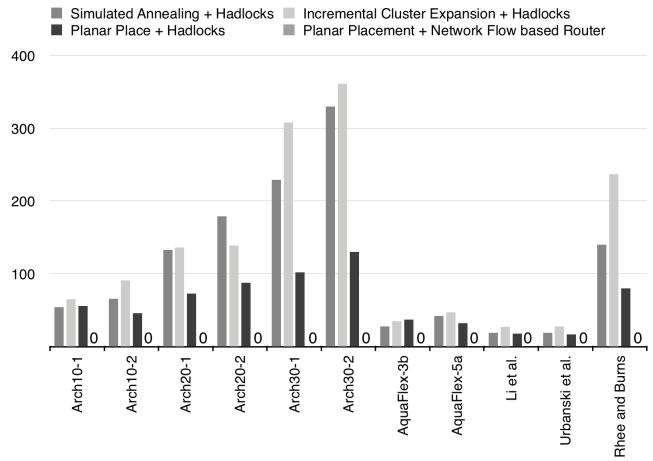


Figure 6.18: The number of intersections for each benchmark with each combination of algorithms. Notice that all the Planar Placement and Network Flow based Router (PP+NFR) benchmarks add zero intersections

Chapter 7

Flow Layer Routing

The flow layer routing of an LoC device lays out the channels that the fluids will flow through during the execution of the experiment. Since there can currently only be one flow layer fabricated on a given chip, there can be now intersections in the final device. Any intersections that do occur need to be replaced with a switch, which can deterministically direct the fluid to the proper components. Additionally the device needs to be connected to external pressure sources to provde the pressure that actually moves the fluids once a valve has been opened. Alternatively peristaltic pumps can be placed on the chip, though typically that would be done prior to the routing phase.

7.1 Maze Routing Algorithms

The first class of routers explored, and the one that has typically been used in the literature, are maze routing algorithms. These algorithms discretize the plan of the LoC device into a grid, where each node is a block that can contain a channel within it. The size of these blocks will be defined as the width of a channel that would flow through it, plus half the width of the spacing required between channels, or a channel and a component. This ensures that the

fabrication specification will obey the design rules as they are specified in the technology file. Each node is connected to the nodes to the North, East, South, and West, as well as diagonally in each of those directions. This grid structure is used in all of the routing algorithms in this thesis.

7.1.1 Hadlock's Maze Routing

Hadlock's maze routing algorithm [36] is an optimization to Lee's maze routing algorithm [54]. In Lee's algorithm, a node adjacent to the port on the source component is selected as the source node. The algorithm then implements an expanding wavefront over a series of iterations. During each iteration, the nodes that are adjacent to any node on the edge of the expanding wavefront are added as sources and marked with the iteration they were discovered at. Once the sink node has been discovered, the algorithm must trace back to the source. The simplest way to trace back is to start at the sink node, and one at a time add each node with decreasing discovery iteration until the source node is added. The path that has traced back will be a shortest path between the source and sink components. The nodes along the path are then labeled as occupied and the process iterates on the next route.

This process unnecessarily explores many nodes that are not on the final path and not even in the same direction as the sink node. The optimization developed by Hadlock implements a directed search. When adding nodes to the wavefront they are added with their manhattan distance to the sink node. This enables a priority queue to be used to direct the search towards nodes that are in the direction of the sink. The direct path between two components is not always available, however, so once the algorithm is no longer able to search in the direction of the sink, it is able to reverse, and get around any obstacles in this way, before continuing the exploration process. This method is a naïve method for routing, but it provides initial results

and has been used in prior works [68]. The next section describes a more intelligent method of performing the flow layer routing.

7.2 Planar Routing Algorithms

7.2.1 Network Flow-Based Router

The network flow-based router does not allow for the introduction of intersections during the routing step. This may limit the number of devices that can be successfully routed, but as added intersections significantly complicate the control layer synthesis step, and frequently render a device too complex to fabricate this limitation is acceptable. Similar to the maze routing algorithms discussed in the previous section, the network flow-based router implements a grid based method.

Routing Grid

A routing grid $R = (U, F)$ is instantiated in the device plane, 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 $m_i \in M$ a vertex u_i for the ports $p_h \in P_i$ is instantiated and added to U. A grid of vertices is then instantiated in the empty space between components. Pseudocode is presented in Fig. 7.1. In lines 17 and 20, edges that represent potential routing channel segments are added to F by instantiating a bidirectional edge f_i with a capacity of 1 between $u_i \in U$ and $u_j \in U$ if and only if $(u_j.x - u_i.x == 1) \oplus (u_j.y - u_i.y == 1)$.

The network flow model ensures that no edge is used more than once, however it is also necessary to ensure that no vertices are used more than once in routing. This is accomplished by splitting each vertex $u_i \in U$ into u_i' and u_i'' and adding a directed edge $f_i = (u_i', u_i'')$ to F. All incoming edges to u_i are now forced through u_i', and all outgoing edges from u_i leave

through u_i''. Thus, any fluid channel that routes through u_i, must go through the edge f_i, and the capacity constraint on edges ensures that there can be at most one such channel using that vertex.

Network Flow Model

Once the grid $R = (U, F)$ has been constructed, the next step is to route channels between the components. This is accomplished using a network flow routing method based on Ref. [87]. Components are processed in-order, and unrouted channels that are incident on each component are routed together using this model. The special nodes super sources, super sinks and sink groups; are added to the routing problem which enables the network flow to simultaneously perform port assignment as well.

Figure 7.2 shows the grid prepared to route $m_i \in M$

- Create a super sink $u_{supersink}$.
- For each $t_j \in T_i$
    1. Add a vertex $u_{sink\_group\_t_j}$ to U.
    2. Add an edge $f_j = (u_{sink\_group\_t_j}, u_{supersink})$ to F with capacity 1 and cost 1.
    3. For each port $p_k \in P_i$ of t_j
        i. Add a vertex u_{p_k} to U.
        ii. Add an edge $f_{p_k} = (u_{sink\_group\_t_j}, u_{p_k})$ to F with capacity 1.
- Create a super source $u_{supersource}$.
- For each port $p_j \in P_i$.
    i. Add a vertex u_{p_j} to U.
    ii. Add an edge $f_{p_j} = (u_{supersource}, u_{p_j})$ to F with capacity 1.

Require: $M :=$ set of all components in the system

Require: $max\_x, max\_y :=$ the maximum x and y values in the plane

Ensure: $R := (U, F)$ grid of vertices

1: **for all** $m_i \in M$ **do**
2: **for all** $p_h \in m_i$ **do**
3: $U \leftarrow U \cup u_i = (p_h.x, p_i.y)$
4: **for all** $0 < x < max\_x$ **do**
5: **for all** $0 < y < max\_y$ **do**
6: **if** $!within\_component(x, y)$ **then**
7: $U \leftarrow U \cup u_i = (x, y)$
8: **for all** $0 < x < max\_x$ **do**
9: **for all** $0 < y < max\_y$ **do**
10: $u_i \leftarrow (x, y)$
11: $F \leftarrow F \cup$ get_east_neighbor(u_i)
12: $F \leftarrow F \cup$ get_south_neighbor(u_i)

Figure 7.1: Grid Creation Algorithm

A set of routes from m_i to all $t_j \in T_i$ is found by computing the maximum flow from $u_{supersource}$ to $u_{supersink}$, followed by a path reclamation step adapted from Lee's algorithm [54]. The paths computed by the network flow algorithm include port assignment at the source and sinks, and may present multiple valid paths. The purpose of the trace back, as shown in Fig. 7.3 is to compute the shortest valid path from the port p_k at each sink t_i to its corresponding port p_j at the source component m_i, as determined by the solution to the network flow problem. The super source, super sink, and sink groups along with their accompanying edges are then removed from the routing grid, and the process repeats for each component in the system, taking care not to route fluid channels that have already been routed.

One problem that may occur is that a route between components m_i and m_j may abut a third component, m_k, potentially blocking one of its ports. To avoid this, we create

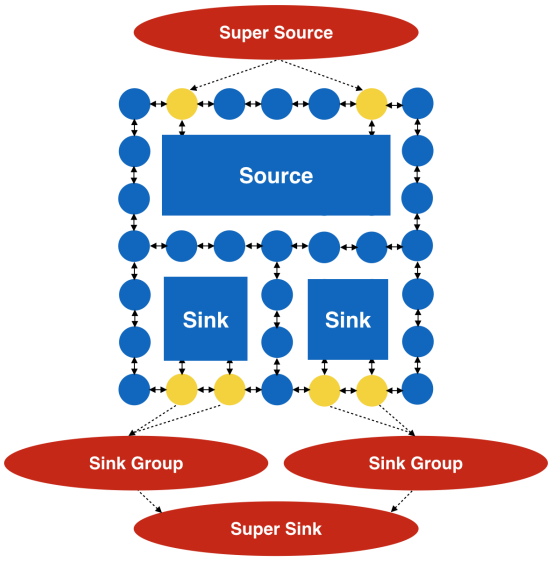


Figure 7.2: The addition of the super source, super sink, and sink group vertices with accompanying edges allows the use of minimum cost maximal flow algorithms to do both port selection and channel routing. Note that a sink group is connected to every port of a particular sink

buffer zones of a few vertices around every component that is not currently being routed. The vertices within the buffer zone are removed from the routing grid ensuring that the ports are not blocked, and only added back to the grid temporarily while routing to or from the respective component.

As fluid channels are routed one-by-one, the routing grid becomes fractured, leading to failures due to routed channel intersections. If a routing failure occurs, the old routes are removed and the queue of components is reordered so that components whose routes have failed in the past are now routed first. The number of times that the component queue may be reordered is limited; if the limit is exceeded, a routing failure is declared. The results of this algorithm are reported in the previous section on the Planar placement algorithm.

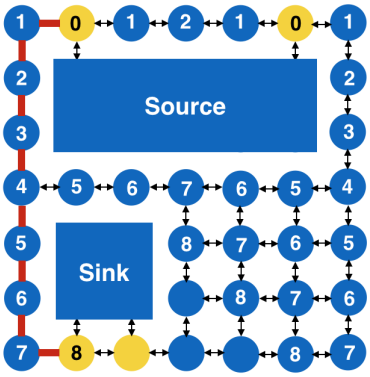


Figure 7.3: The maximum flow minimum cost network flow algorithm numbers the nodes as they are discovered, ending when it reaches an unused port. These numbers are then used to trace back the path of the channel route.

Chapter 8

Fluid Routing

8.1 Fluid Routing Without Washing

The input to the fluid router is a sequencing graph $A = (Op, \mathcal{E})$, shown in Fig. 8.1, and the architecture $D = (V, C)$, as shown in Fig. 8.2. Where $Op \in A$ is the set of operations to be performed, $\mathcal{E} \in A$ defines the dependencies between the operations, $V \in D$ are the components that perform the operations, and $C \in D$ represents the channels between components.

Each $e_{ij} \in \mathcal{E}$ is a fluid that needs to be routed from the source $\mathcal{B}(op_i) = m_i$ to the sink $\mathcal{B}(op_j) = m_j$, where $m_i, m_j \in V$ are the components that execute $op_i, op_j \in Op$, as defined by the binding function $\mathcal{B}$. The fluid e_{ij} needs to be routed before the operation op_j can be executed.

The objective of the fluid routing is to determine the set of disjoint paths $P = \{p_1, p_2, \ldots p_n\}$ that need to be routed during each time step such that all of the fluids will be routed. Each path $p_i \in P$ is a three part path in D:(i) a sub-path from a buffer input to the source, s_i; (ii) a sub-path from the source s_i to the sink, t_i; and (iii) a sub-path from t_i to a fluid output.The constraints on the fluid routing are as follows:

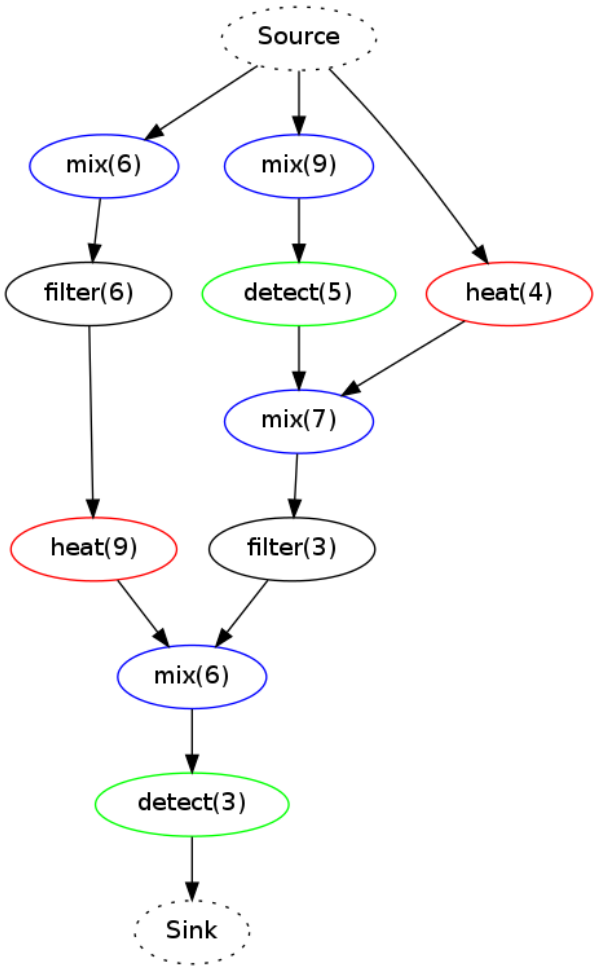


Figure 8.1: Sequencing graph representation of the synthetic benchmark with 10 operations from [2]

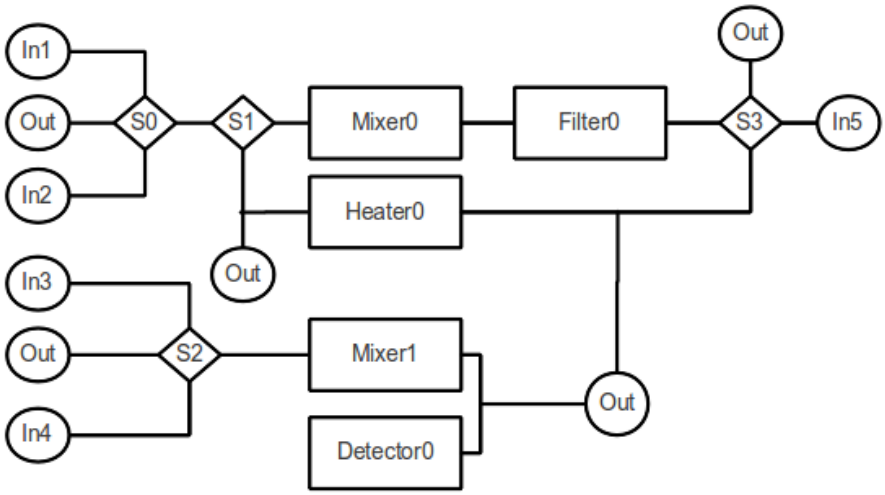


Figure 8.2: Functional model of a chip to execute the assay shown in Fig. 8.1. The circles indicate I/O punches, the rectangles indicate the components, and the lines represent the interconnections between them. The diamonds correspond to switches on the control layer.

- The partial ordering defined in the sequencing graph must not be violated,
- there may not be any cross contamination,
- and accidental dilution of the fluids must be avoided.

It is important, here, to recognize that simply opening microvalves along the path does not actuate fluid motion; pressurized fluid must be injected from an external source in order to cause fluid to flow.

The paths in P will all be routed in parallel, therefor the time required is equal to the time required to route the longest path, $L(P)$. Let $l(p_i)$ be the length of path p_i, then the time required to route the set P is proportional to $L(P) = max\{l(p_1), l(p_2), \ldots, l(p_n)\}$. This problem is a straight forward variant of the *Disjoint Path Problem* which was one of the original NP-complete problems identified by Karp [50].

8.2 Naïve Method

The naïve fluid routing algorithm takes in as input, a list of movements which must be executed with respect to a partial order. The algorithms, shown in Fig. 8.3 will iterate through the different movements while maintaining the order, and will continue to route fluids until the assay completes, or becomes unroutable. The algorithm takes place in the following four phases:

Phase 1 (Execute Operations): First the operations that are ready, all input fluids have arrived, are executed. These operations will be executed in parallel, and so the timestep is advanced to the completion of the longest operation. All of the operations executed during these step will be marked as completed. Initially only the fluid input operations are ready.

Phase 2 (Determine Ready Movements): A movement is said to be ready when the action that produces that fluid has completed. Additionally any movements that were delayed from a previous routing step will be added to the list of ready movements.

Phase 3 (Compute Routes): Fluids are routed one-by-one and evaluated to ensure that legal routing has taken place. The routing phase occurs by calling Dijkstra's algorithm three times. First the buffer input is routed to the source s_i, then from s_i to t_i, and finally from t_i to a fluid output. If successful then the fluid is allowed to follow the path found by Dijkstra's algorithm; if not, the fluid is not allowed to route and will be delayed to future iterations; partial routes are not allowed. Once a route has been verified, the vertices and edges along the route are contaminated and therefore marked invalidated. An invalidated node is essentially removed from the graph, and will be inserted back into the graph once it has been washed. This will prevent cross contamination between the samples during

Require: Operations with a partial ordering

Ensure: Schedule of operations {M}aïve Method

```
readyOperations ← ∅
readyMoves ← ∅
delayedMoves ← ∅
timeStep ← 0
while operations remain unscheduled do
    for all op_i ∈ operations do
        if op_i.inputs completed then
            readyOperations ← op_i
            readyMoves ← op_i.output
    for all op_i ∈ readyOperations do
        Execute op_i
        Schedule op_i
    timeStep ← next event
    for all move_i ∈ readyMoves do
        Dijkstra(bufferInput, move_i.source)
        Dijkstra(move_i.source, move_i.sink)
        Dijkstra(move_i.sink, Waste)
        if Routing succeeds then
            Schedule move_i
            Set move_i as completed
        else
            delayedMoves ← move_i
    timeStep ← next event
    Call Washing()
    timeStep ← end of washing
    readyMoves ← delayedMoves
    delayedMoves ← ∅
```

Figure 8.3: Naïve fluid routing method

the execution of the assay. Figure 8.4 shows two independent routing steps, the routes in each step are clearly disjoint. There would need to be a washing step between these steps.

Phase 4 (Wash): This phase will wash the contaminated nodes and insert them back into the graph, allowing them to be used for subsequent routes. If the device is unable to be washed, routes will be removed until the washing can complete. This process is discussed in Section 8.3.

The four phases repeat until the assay has complete, or it is determined that a routing failure has occurred and the assay cannot be completed as described on the given device. If on any iteration after a wash no additional fluids are able to route, this will indicate a routing failure and the algorithm will abort.

These four phases lead to a discretization of the assay into operation, movement, and wash steps, as shown in Fig. 8.5, by advancing the timestep to the next *event*. In phase 1 an operation step is set up, and the *event* is defined as the end of the longest operation to complete. The *event* for phase 2 and 3 is defined as the end of the longest route scheduled. Finally the timestep is advanced to the end of the wash step in phase 4.

This discretization of the assay leads to wasted time steps. If a short operation is executing in parallel with a much longer operation during phase 1 then the movement produced by the short operation will need to wait until the end of the step before it is able to route. This will delay the movement of the fluid, as well as any operations that follow that movement.

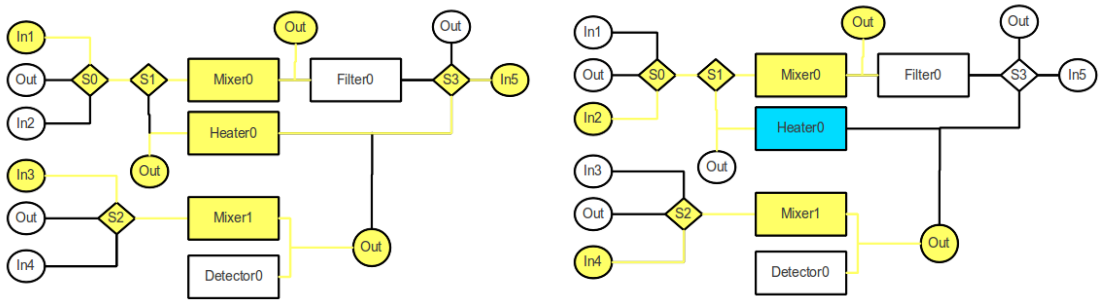


Figure 8.4: The first (a) and second (b) routing steps for the assay from Fig. 8.1 executing on the chip from Fig. 8.2. These two steps would need to be broken up by a washing step, as shown in Fig. 8.5. The light yellow shows the components that are currently being used for moves. Heater0 is being used during (b) to run an operation concurrently with the moves being executed.

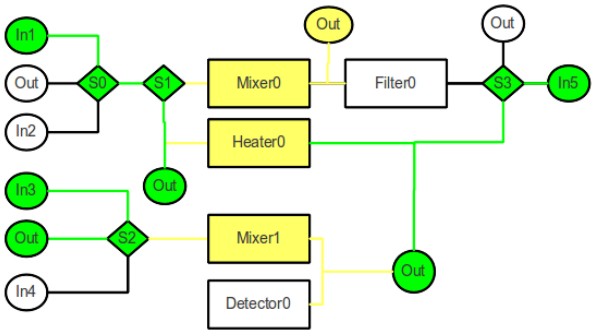


Figure 8.5: This illustrates the washing step that would occur between Fig. 8.4(a) and (b). The contaminated nodes are washed, without diluting the reagents that remain in the components. The green indicates the nodes that are washed, while the light yellow indicates the nodes that remain contaminated.

8.3 Combined Scheduler and Router

The combined scheduler and router (CSR) algorithm will more tightly integrate the scheduling of operations with the routing of the fluids. The algorithm extends the naïve method to allow for operations to be scheduled concurrently with either wash steps or movements. This removes the discretization of the assay which led to wasted time.

This is done by redefining what is classified as an *event* from the naïve method. Originally it an *event* would be the end of any given step, however, as shown above, this leads to wasted time steps and under utilization of the device. To create a tighter schedule, an *event* is defined to be end time of the movement or operation that completes soonest, not last. This *event* triggered algorithm requires several adjustments to how ready operations, shown in Fig. 8.6, and movements are handled. Instead of marking them as complete and advancing the time step, they are added to a list of initiated operations/movements and add them to the schedule once they have completed.

These minor changes enable finer grained control over the current time step, allowing it to advance much more incrementally providing a tighter integration of the routing of fluids and scheduling of operations. This fine grained control is what allows the wasted time steps to be removed from the naïve method, as shown in Fig. 8.4(b).

8.4 Washing the LoC

Unlike traditional VLSI devices, LoC's have to consider cross contamination between different fluids within the device. If cross-contamination occurs, the entire assay could be invalidated and need to be run again. Virtually all microvalve-based LoCs require washing at some point during the execution of a biological protocol. Washing the LoC decontaminates

Require: Operations with a partial ordering

Ensure: Schedule of operations {C}SR Fluid Routing Phase 1

```
initiatedOperations ← ∅
initiatedMoves ← ∅
readyOperations ← ∅
readyMoves ← ∅
delayedMoves ← ∅
for all op_i ∈ initiatedOperations do
    if op_i completed then
        readyMoves ← op_i.output
        Schedule op_i
        Remove op_i from initiatedOperations
for all move_i ∈ initiatedMoves do
    if move_i completed then
        Schedule move_i
        Remove move_i from initiatedMoves
for all op_i ∈ operations do
    if op_i.inputs completed then
        initiatedOperations ← op_i
```

Figure 8.6: CSR Algorithm

components and microchannels to facilitate contamination-free routing of future fluids. The objective of each washing step is to decontaminate as many components and microchannels as possible by injecting buffer into the LoC and routing it through contaminated regions, but without diluting any of the fluids presently stored on the chip.

Washing is achieved similarly to the routing of the fluids in the system. The primary goal behind the washing step is to decontaminate all nodes that have been contaminated due to the previous routing step. The entire washing process takes place in four different phases:

Phase 1 (Buffer Inputs to Source Nodes): it is guaranteed that the only nodes that are contaminated fall along the previously computed routes, $p_i = \{v_1, v_2, \ldots, v_n\}$, therefore flooding the system to perform the washing is unnecessary. Instead, Dijkstras algorithm is run using the buffer input (B) as the source node, and $p_i.source = v_1$ of the path as the sink.

Phase 2 (Trace Routes): Only contaminated components are being washed, so it is sufficient to trace through the routes which the fluids have flowed through, as previously computed, in order to remove the contamination. The one caveat to keep in mind here is that all the destination nodes for each route still contain fluid. To avoid diluting the solution, each path p_i is only traced to the second to last node, $\{v_1, v_2, \ldots, v_{n1}\}$.

Phase 3 (Buffer Disposal): Disposing of the buffer fluid is similar to the process of phase 1. Dijkstras algorithm is employed again, using the second to last node, v_{n1} in each path as the source node(s), and the waste output W as the sink node. The difference between the Dijkstras used during washing, and the one used during fluid routing is that the nodes encountered during the washing iterations of Dijkstras are not invalidated, allowing the routes in each individual phase of washing to use nodes on routes from the previous

phases. Once the routes have completed, the nodes visited are marked validated again in the graph, and can be used in subsequent fluid routing iterations.

Phase 4 (Fail safe): If any of the previous 3 phases fails to route, then the washing has failed. In this case, previously computed routes are removed and washing is attempted again until the wash step is able to complete. This will require more washing steps and more routing steps which will extend the the execution time of the assay but it will prevent the chip from becoming unroutable due to contamination.

Due to the underlying mechanisms of pressurized control, it is not possible to drain buffer from the chip entirely to leave it empty; however, by definition, buffer is chosen to be non-reactive with the other fluids that comprise the assay. Therefore, the process of opening new paths to transport fluid will push part of the buffer that remains in the chip after washing out of the chip once the process completes.

8.5 Results

The fluid routing algorithms were run on a set of publicly available synthetic benchmarks [2] as well as several real life benchmarks: PCR [2] and Chromatin Immunoprecipitation (ChIP) [88]. The results of the algorithms are presented in Table 8.1. The table shows the execution time of the assays, δ_G , with and without washing. The execution times are shown using the naïve method, Combined Scheduler and Router (CSR), and the full path enumeration from Minhass et al. [68]. The results from the method from [68] are not shown for the assay run with washing since they do not account for washing in their algorithms.

It is clear from the table that the CSR algorithm out-performs the naïve method for every benchmark, averaging a speedup of 1.8x. The CSR algorithm performs much better on

Table 8.1: Fluid Routing

Benchmark		δ_G w/Washing			δ_G w/out Washing			Run Time	
		naïve	CSR	[68]	naïve	CSR	[68]	naïve	CSR
Synthetic	10	52 s	33 s	N/A	45 s	33 s	35.7 s	2 s	2 s
	20	79 s	51 s	N/A	62 s	49 s	N/A	6 s	8 s
	30	112 s	67 s	N/A	82 s	65 s	46.1 s	17 s	20 s
	40	148 s	104 s	N/A	100 s	98 s	59.5 s	34 s	47 s
	50	176 s	131 s	N/A	114 s	129 s	60.4 s	45 s	72 s
Real Life	PCR	32 s	21 s	N/A	27 s	21 s	19.7 s	3 s	2 s
	ChIP	31420 s	9081 s	N/A	31399 s	9081 s	N/A	23 s	31 s

highly parallelizable assays, having as much as a 3.5x speedup in execution time for the ChIP benchmark with and without washing. The run times for the algorithms are presented in the last two columns of the table. The run times for [68] are unavailable. The naïve method performs faster in general, but the improvement on the schedules from the CSR approach outweighs the time lost during the routing process.

The full enumeration algorithm does outperform both algorithms when washing is not taken into account, however the focus of these algorithms was to introduce washing and produce realistic assay schedules. The schedules produced by Minhass et. al [68] could not realistically be executed without contaminating the components and therefore nullifying the results. Additionally their schedules are produced by fully enumerating all of the paths and their conflicts, and using them to produce a schedule. This approach is not scalable to large chips. The details of their algorithm are also insufficient for it to be entirely clear how decisions are made between multiple paths, making a direct comparison difficult.

It is important to note here that the results for both algorithms have been obtained using the same chip placements, however access to the placed chips from [68] is unavailable. The difference in routing times could also be partially accounted for by a difference in placements.

8.6 Conclusion

A novel fluid routing algorithm which routes the fluids without enumerating every possible route, which is not scalable to larger designs. Additionally it is the first ever fluid routing algorithm that will insert the necessary washing steps to prevent cross contaminations. Without these washing steps the results would become contaminated and the chip, and the results would become useless. The naïve fluid routing algorithm is also extended to integrate the scheduling of the operations with the routing of the fluids. This integration produces a more tightly coupled schedule, reducing wasted time which leads to a shorter execution times across all benchmarks we tested.

The CSR algorithm led to an average of a 1.8x speed up in execution time of the assay, with a speedup of 3.5x on the highly parallelizable ChIP assay. Microfluidic chips are frequently used to execute highly parallelizable assays to increase throughput, making the speedup on the ChIP assay quite significant.

Part IV

Control Layer

Chapter 9

Control Layer Synthesis and Optimization

The synthesis of the control layer involves the binding of valves to control channels, the allocation of external pressure sources, and the subsequent binding of control channels to an external pressure source. The external pressure sources will provide the pressure to actuate the valves thereby controlling the flow. When an external pressure source is actuated (1) then the valve(s) connected to that channel, control line, will be closed, when it is not actuated (0) the valves will be open. The number of these external pressure sources that will fit on a device is limited, and is one of the most limiting factors in LoC device design. *According to Quake's foundry design rules [3] external punch holes are limited to thirty-five, this number may be flexible.*

The algorithms presented here will perform control line minimization, and as a result minimize the number of external pressure sources, the pressure sources will then be allocated, and then during the routing of the control channels, they will be bound to specific external

pressure sources. If the control line minimizations is unable to reduce the number of control lines to below the maximum allowed, synthesis will fail.

Problem Description: A set of valves, $v_i \in V$, is given along with their actuation sequences, as_i, and an external pressure sources constraint, MAX_P. An actuation sequence as_i is defined by a sequence of 1 (closed), 0 (open), or x (don't care), that describes the actuation status of the valve v_i at each time step. The number of control lines, c_j, needed to control all $v_i \in V$, must be minimized such that $j <= MAX\_P$ and no valve is open when it should be closed, or vice versa.

Note: A valve with the actuation status of x (don't care) indicates that the status of the valve will have no affect whatsoever on the flow of fluid through the device.

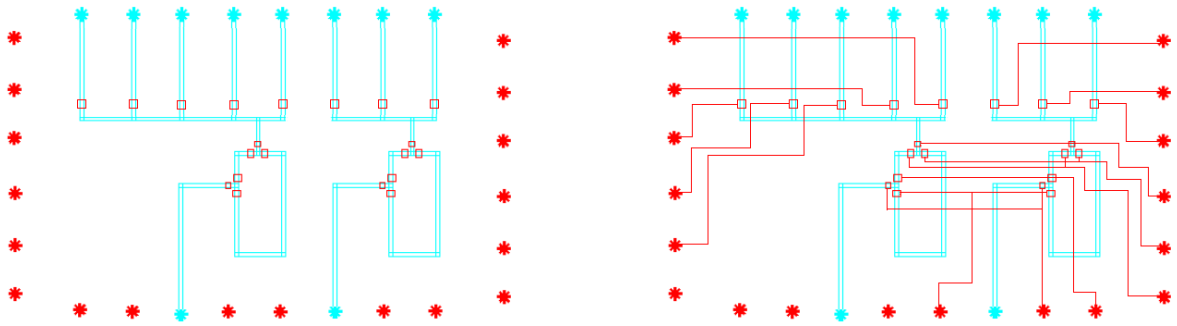
9.0.1 Graph Coloring Method

The graph coloring method described in [68] is employed to perform the optimization of the external pressure sources. The graph is constructed by first adding a node for each valve on the chip, the nodes are added with knowledge of their spatial location on the chip. This allows for decisions on coloring to be made later on. An edge is then added between any two nodes whose actuation sequences differ during at least one time step. An actuation sequence is said to differ at a time step if one actuation sequence has a 1, while the other has a 0. If either actuation sequence has a don't care (x) then the actuation sequences do not differ at that time step. The edges indicate valves that cannot share a control line. A general graph coloring solver is used to identify which valves can share control lines, the nodes (valves) which share the same color can be on the same control line. The full algorithm is shown in Fig. 9.3.

Once the graph is colored the control lines are ready to be routed. It may end up that some of the sets of valves are difficult, or even impossible to route. To solve this the valves will

Valve No.	Time Steps (s)						Color
	0	2	4	5	8	...	
1	0	0	X	X	0	...	-
2	0	0	1	1	0	...	Color - 0
3	0	1	0	0	1	...	Color - 11
4	1	0	0	0	0	...	Color - 1
5	0	1	0	0	1	...	Color - 11
6	0	0	1	1	0	...	Color - 0
7	1	0	0	0	0	...	Color - 1
8	0	0	Mix	Mix	0	...	Color - 14
9	0	0	Mix	Mix	0	...	Color - 8
10	0	0	Mix	Mix	0	...	Color - 2
...	...	...	...	...	...	...	...
27	0	0	X	X	X	...	-
28	0	0	1	1	1	...	Color - 3
29	0	1	0	0	0	...	Color - 6
30	1	0	0	0	0	...	Color - 1
31	0	1	0	0	0	...	Color - 6
32	0	0	1	1	1	...	Color - 3
33	1	0	0	0	0	...	Color - 1
34	0	0	Mix	Mix	Mix	...	Color - 9
35	0	0	Mix	Mix	Mix	...	Color - 4
36	0	0	Mix	Mix	Mix	...	Color - 7
...	...	...	...	...	...	...	...
42	0	X	X	X	X	...	-
43	0	1	1	X	X	...	Color - 13
44	0	1	1	X	X	...	Color - 13
...	...	...	...	...	...	...	...
89	X	X	X	X	X	...	Color - 4

Figure 9.1: Example actuation sequences for valves from [68].



(a) (b)

Figure 9.2: (a) An example chip with the valves and external pressure sources bound. It is clear from the image that without control line sharing this chip would be un-route-able, as there are more valves than external pressure sources. (b) Shows the example device with the control lines routed, manually, and some lines are shared so that the chip is route-able.

be dynamically selectable based on their difficulty to route, and moved from the valve set it is contained in, to a new valve set. The graph used for the coloring is maintained, with a few updates; the graph is essentially inverted, creating an edge between each set of nodes that did not have an edge in the original graph, and removing all others. These edges will indicate nodes that the selected node *can* share a control line with. Additionally these edges will be weighted with the Manhattan distance between the valves corresponding to the nodes at the end points.

This updated graph can be used to dynamically adjust the color of a node when it's corresponding valve is difficult to route. After a route fails, the color of the set of valves that are deemed un-route-able, x, is taken. In this set the valve that is most difficult to route is selected, call it v_i. The nearest adjacent valve to v_i with a different color y such that $y \neq x$ is chosen. This is done using the edges weighted with Manhattan distance. The valve, v_i is then recolored with color y and routing is attempted again. The case may arise where a node is continually swapped between two colors, say valve v_i has color x on the first attempt, color y

on the second attempt, and then back to color x on the third attempt, and so on. To prevent this, a list of the colors a node has already used is maintained. If any node exhausts all of the colors available to it the graph is deemed un-route-able.

Require: V: Set of Valves.

MAX_P: Maximum number of external pressure sources

Ensure: A colored graph indicating the valves that can share control lines. Nodes of the same color can share a control line.

Construct a node, n_i, for each valve in the biochip.

for all $v_i, v_j \in V$ **do**

if $as_i \neq as_j$ **then**

Construct an edge, e_{ij}, between n_i and n_j.

Perform Graph Coloring on the constructed graph using tabu graph coloring from [68].

Figure 9.3: Graph Coloring algorithm for control synthesis problem.

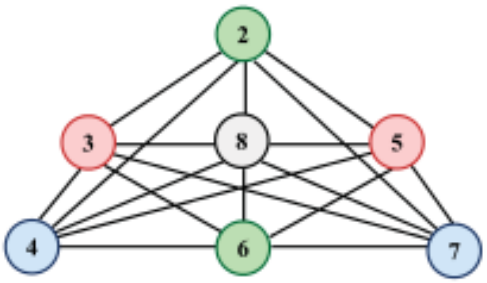


Figure 9.4: Graph coloring example from [68]

9.1 Control Layer Routing

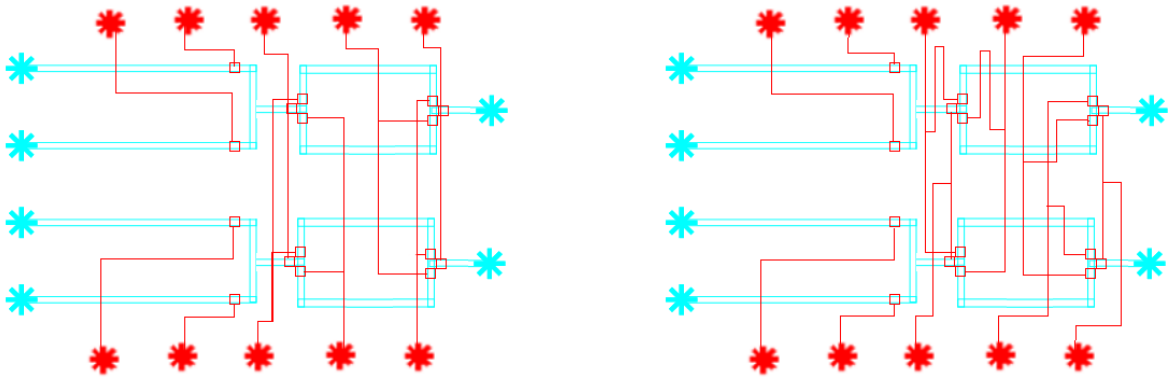
The control synthesis step allocated the necessary valves and placed them deterministically based on the flow layer placement. The number of external control lines (pressure sources), was then (optionally) optimized during control layer optimization and the external pressure sources were allocated. *The external pressure sources are assumed to be placed initially.*

The control channels must be routed from the external pressure sources to the valves they control. This will require either routing to (1) a single valve or (2) routing to multiple valves which are sharing the same external pressure source. In case (2), the shared valves must be tethered together and routed to the external pressure source. Case (1) is a simplified version which does not require any valves to be tethered together. Figure 9.5a shows an example routing using shared control lines, the routing was performed manually.

Problem Definition: Given a set of $v_i \in IV \subset V$ independent channel valves and a set $sv_i \in SV \subset V$ of Shared Valve lines where each sv_i is a set of valves $v_j \in sv_i$ that share the same control line. The set $IV \cap SV = \emptyset$ and $IV \cup SV = V$ with V being the set of all valves. Additionally a set $p_i \in EP$ of possible locations for External Pressure sources is given. A routing must be provided such that each valve $v_i \in IV$ and shared valve set $sv_i \in SV$ are routed successfully to an independent $p_i \in EP$ while minimizing overall length $L = \sum_{\forall v_i \in V} l(v_i)$.

9.1.1 Unconstrained Control Line Routing

The unconstrained control line routing problem is equivalent to the *Steiner tree problem*, or routing from a single source to multiple sinks.A control line must be routed from each source to the corresponding sink(s), while minimizing overall length. There is no constraint on the order in which the sink(s) (valves) are reached, actuation time, or the skew between the



(a) (b)

Figure 9.5: An example chip with the valves and external pressure sources placed and the channels between them routed (a) with and (b) without using equal length routing techniques.

actuation times of the valves. The lines with a single valve for the sink is just a simplified case of this problem and will be handled using the same method. The routing will be performed using a modified Hadlock's routing algorithm to route to mulitple sinks.

9.1.2 Constrained Control Line Routing

In the constrained control line routing problem, the constraint that the control signal needs to arrive at the same time for all valves on a single shared control line is added. This problem is similar to the clock signal routing problem in electronic systems. In the clock signal routing problem, any clock source that are adjacent (they are only separated by combinational logic) will need to have a clock arrival time within a prescribed skew.

The Delayed Merge Embedding (DME) algorithm from [49] will be utilized to perform a Zero Skew Tree (ZST) routing to these valves. First merging segments are determined between

each pair of valves. This merging segment indicates a reasonable selection of midpoints between two valves where the route to each valve will be of equal length. Once the merging segments have been determine, the algorithm works back up from the leaf nodes (valves) and select the points along the midsegments to use for the routing. The generated tree provides equal length straight line routing to all valves along the shared channel. If there were only one tree this would now be done, however there are multiple such trees, one for each shared control line, on the device. The next step is to route all of these paths while respecting the other paths on the device. Hadlock's algorithm is employed again to route these channels. Hadlock's router does not take into consideration the equal length constraint of the routes, and so the skew may be enlarged.

Yao et. al implemented the control layer routing method using a negotiated congestion router in place of the Hadlock's router in PACOR [92]. In their work a post processing step is used to determine if a route does not meet the equal length criteria. That route is then ripped up and rerouted with a detour embedded to match the equal length criteria.

Part V

Conclusion

9.2 Conclusion

This thesis presents an end to end tool chain that is capable of taking a high level language description of either a biochemical application, or an LoC device, and producing a design that is capable of being fabricated. The three major tracks that this system could take starts with (i) just a biochemical application (ii) just an LoC device specification (iii) a biochemical application and an LoC device specification.

Biochemical application only is ideal for a scientist that has an application in mind and would like to develop a new LoC to perform that experiment. This scientist need not have the expertise in fabrication technologies or even necessarily in microfluidics. The biochemical application, or assay, be written in a high level language like biocoder, can be compiled and a sequencing graph representing the assay generated. This sequencing graph is then used to allocate the necessary components to complete the assay, and extract the required connections between those components, defining a netlist for an LoC device. This netlist can then be synthesized into a design that can be fabricated, as described in the following section.

LoC device specification only allows an engineer to design novel devices or entities. These device or entities can be designed to address a very specific operation, or generally applicable to a class of operations. The LoC netlist can be described in the high level microfluidic hardware design language (mHDL). This netlist describes the flow layer, and the control layer is derived from the entity library files. The flow layer is then placed and routed, the control layer synthesized, and finally the control layer is routed. Without a specific application in mind, the control layer will not be able to be optimized as the actuation sequence of the valves is unknown. The output is a description of the LoC in

an IR, allowing it to be used by scientists that need the functionality, or in an STL file which can be used to fabricate the LoC.

Biochemical application and LoC specification is envisioned to be how many scientists and engineers working together will be able to use the system. In this case there will be a library of available LoC architectures that has been generated by the engineers. The scientists are then able to define the biochemical application, and attempt to map it onto existing devices. This allows labs to have a stock pile of existing devices, and not have to redesign a new device for every application that is developed. The architecture already has a netlist, and is already placed and routed. The biochemical application is then scheduled onto this architecture, assuming it has the necessary components and connections. The algorithms developed in this thesis are capable of this step, although if it is unable to complete the scheduling process it will declare a failure.

Bibliography

[1] http://openwetware.org/wiki/Main_Page.

[2] https://sites.google.com/site/mlsibiochips/.

[3] http://web.stanford.edu/group/foundry/.

[4] http://www.boost.org/.

[5] www.Illumina.com.

[6] Ahmed M. Amin, Raviraj Thakur, Seth Madren, Han-Sheng Chuang, Mithuna Thottethodi, T. N. Vijaykumar, Steven T. Wereley, and Stephen C. Jacobson. Software-programmable continuous-flow multi-purpose lab-on-a-chip. *Microfluidics and Nanofluidics*, apr 2013.

[7] Ahmed M. Amin, Mithuna Thottethodi, T. N. Vijaykumar, Steven T. Wereley, and Stephen C. Jacobson. AquaCore : A Programmable Architecture for Microfluidics.

[8] Ahmed M. Amin, Mithuna Thottethodi, T. N. Vijaykumar, Steven T. Wereley, and Stephen C. Jacobson. Automatic Volume Management for Programmable Microfluidics. In *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation PLDI 08*, volume M of *PLDI '08*, pages 56–67. ACM Press, 2008.

[9] Nada Amin. *Computer-Aided Design for Multilayer Microfluidic Chips by Nada Amin.* Msc, Massachussett's Institute of Technology, 2009.

[10] Nada Amin, William Thies, and Saman Amarasinghe. Computer-aided design for microfluidic chips based on multilayer soft lithography. pages 2–9, 2009.

[11] Vaishnavi Ananthanarayanan and William Thies. Biocoder: A programming language for standardizing and automating biology protocols. *Journal of biological engineering*, 4(1):13, jan 2010.

[12] Ismail Emre Araci and Philip Brisk. Recent developments in microfluidic large scale integration. *Current Opinion in Biotechnology*, 25:60–68, feb 2014.

[13] Ismail Emre Araci and Stephen R. Quake. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. *Lab on a Chip*, 12(16):2803–2806, aug 2012.

[14] Frederick Kiguli Balagadd. *Microfluidic technologies for continuous culture and genetic circuit characterization.* Doctor of philosophy, California Institute of Technology, 2007.

[15] Frederick K Balagaddé, Lingchong You, Carl L Hansen, Frances H Arnold, and Stephen R. Quake. Long-term monitoring of bacteria undergoing programmed population control in a microchemostat. *Science (New York, N.Y.)*, 309(5731):137–40, jul 2005.

[16] John M Boyer and Wendy J Myrvold. On the Cutting Edge : Simplified O (n) Planarity by Edge Addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[17] Krishnendu Chakrabarty and Tao Xu. *Digital Microfluidic Biochips: Design Automation and Optimization*. CRC Press, 2010.

[18] Krishnendu Chakrabarty and Jun Zeng. Design automation for microfluidics-based biochips. *ACM Journal on Emerging Technologies in Computing Systems*, 1(3):186–223, 2005.

[19] Curtis D Chin, Tassaneewan Laksanasopin, Yuk Kee Cheung, David Steinmiller, Vincent Linder, Hesam Parsa, Jennifer Wang, Hannah Moore, Robert Rouse, Gisele Umviligihozo, Etienne Karita, Lambert Mwambarangwe, Sarah L Braunstein, Janneke van de Wijgert, Ruben Sahabo, Jessica E Justman, Wafaa El-Sadr, and Samuel K Sia. Microfluidics-based diagnostics of infectious diseases in the developing world. *Nature medicine*, 17(8):1015–9, aug 2011.

[20] Usha Chitkara, Louanne Hudgins, H Christina Fan, Yair J Blumenfeld, Usha Chitkara, Louanne Hudgins, and Stephen R. Quake. Noninvasive diagnosis of fetal aneuploidy by shotgun sequencing DNA from maternal blood. *Proceedings of the National Academy of Sciences of the United States of America*, 105(42):16266–71, oct 2008.

[21] Hou-Pu Chou, Marc Alexander Unger, and Stephen R. Quake. A microfabricated rotary pump. *Biomedical Microdevices*, 3(4):323–330, 2001.

[22] Marek Chrobak and Thomas H Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54:241–246, 1995.

[23] Udara Dharmasiri, MA Witek, AA Adams, JK Osiri, ML Hupert, TS Bianchi, DL Roelke, and SA Soper. Enrichment and detection of Escherichia coli O157: H7 from water samples using an antibody modified microfluidic chip. *Analytical Chemistry*, 82(7):2844–2849, 2010.

[24] Trung Anh Dinh, Tsung-yi Ho, Shigeru Yamashita, and Yuko Hara-Azumi. A Clique-Based Approach to Find Binding and Scheduling Result in Flow-Based Microfluidic Biochips. pages 199–204.

[25] Philip Nathanael Duncan, Siavash Ahrar, and Elliot E Hui. Scaling of Pneumatic Digital Logic Circuits. *Lab on a Chip*, jan 2015.

[26] Shirit Einav, Doron Gerber, Paul D. Bryson, Ella H. Sklan, Menashe Elazar, Sebastian J. Maerkl, Jeffrey S. Glenn, and Stephen R. Quake. Discovery of a hepatitis C target and its pharmacological inhibitors by microfluidic affinity analysis. *Nature biotechnology*, 26(9):1019–27, sep 2008.

[27] D. Falconnet, A. Niemisto, R.J. Taylor, M. Ricicova, T. Galitski, I. Shmulevich, and Carl L Hansen. High-throughput tracking of single yeast cells in a microfluidic imaging matrix. *NIH Public Access*, 11(3):466–473, 2012.

[28] Cong Fang, Yanju Wang, Nam T. Vu, Wei Yu Lin, Yao Te Hsieh, Liudmilla Rubbi, Michael E. Phelps, Markus Müschen, Yong Mi Kim, Arion F. Chatziioannou, Hsian Rong Tseng, and Thomas G. Graeber. Integrated microfluidic and imaging platform for a kinase activity radioassay to analyze minute patient cancer samples. *Cancer Research*, 70(21):8299–8308, 2010.

[29] L M Fidalgo and Sebastian J Maerkl. A software-programmable microfluidic device for automated biology. *Lab on a Chip Miniaturisation for Chemistry and Biology*, 11(9):1612–1619, may 2011.

[30] Rongke Gao, Juhui Ko, Kiweon Cha, Jun Ho Jeon, Gi Eun Rhie, Jonghoon Choi, Andrew J. DeMello, and Jaebum Choo. Fast and sensitive detection of an anthrax biomarker using SERS-based solenoid microfluidic sensor. *Biosensors and Bioelectronics*, 72:230–236, 2015.

[31] Daniel Grissom, Christopher Curtis, Skyler Windh, Calvin Phung, Navin Kumar, Zachary Zimmerman, Kenneth ONeal, Jeffrey McDaniel, Nick Liao, and Philip Brisk. An open-source compiler and PCB synthesis tool for digital microfluidic biochips. *Integration, the VLSI Journal*, pages 1–25, apr 2015.

[32] Daniel Grissom, Jeffrey McDaniel, and Philip Brisk. Performance and cost analysis of NoC-inspired virtual topologies for digital microfluidic biochips. *2014 International Symposium on Integrated Circuits (ISIC)*, pages 352–355, dec 2014.

[33] Daniel T Grissom. *Design of Topologies for Interpreting Assays on Digital Microfluidic Biochips A Dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science*. PhD thesis, University of California, Riverside, 2014.

[34] Daniel T Grissom, Jeffrey McDaniel, and Philip Brisk. A Low-cost Field-Programmable Pin-Constrained Digital Microfluidic Biochip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(11):1657–1670, 2014.

[35] William H. Grover, Alison M Skelley, Chung N Liu, Eric T Lagally, and Richard a Mathies. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. *Sensors and Actuators B: Chemical*, 89(3):315–323, apr 2003.

[36] F. O. Hadlock. A shortest path algorithm for grid graphs. *Networks*, 7(4):323–334, 1977.

[37] Carl L Hansen, Morten O A Sommer, and Stephen R. Quake. Systematic investigation of protein phase behavior with a microfluidic formulator. *Proceedings of the National Academy of Sciences of the United States of America*, 101(40):14431–6, oct 2004.

[38] Jong Wook Hong, Yan Chen, W French Anderson, and Stephen R. Quake. Molecular biology on a microfluidic chip. *Journal of Physics: Condensed Matter*, 18(18):S691–S701, 2006.

[39] Jong Wook Hong and Stephen R. Quake. Integrated nanoliter systems. *Nature biotechnology*, 21(10):1179–83, oct 2003.

[40] Jong Wook Hong, Vincent Studer, Giao Hang, W French Anderson, and Stephen R. Quake. A nanoliter-scale nucleic acid processor with parallel architecture. *Nature biotechnology*, 22(4):435–439, 2004.

[41] Martin Simonsen Hørslev-petersen and Thomas Onstrup Risager. Routing Algorithms for Flow-based Microfluidic Very Large Scale Integration Biochips. (June), 2013.

[42] Kai Hu, Trung Anh Dinh, Tsung-Yi Ho, and Krishnendu Chakrabarty. Control-Layer Optimization for Flow- Based mVLSI Microfluidic Biochips. In *CASES*, number June, 2014.

[43] Kai Hu, Tsung-Yi Ho, and Krishnendu Chakrabarty. Testing of flow-based microfluidic biochips. In *2013 IEEE 31st VLSI Test Symposium (VTS)*, pages 1–6. Ieee, apr 2013.

[44] Kai Hu, Tsung-Yi Ho, and Krishnendu Chakrabarty. Test generation and design-for-testability for flow-based mVLSI microfluidic biochips. In *2014 IEEE 32nd VLSI Test Symposium (VTS)*, pages 1–6. Ieee, apr 2014.

[45] Erik C. Jensen, William H. Grover, and Richard A. Mathies. Micropneumatic digital logic structures for integrated microdevice computation and control. *Journal of Microelectromechanical Systems*, 16(6):1378–1385, 2007.

[46] David S Johnson, Cecilia R Aragon, Lyle A Mcgeoch, and Catherine Schevon. Optimization by Simulated Annealing: An Experimental Evaluation: Part II, Graph Coloring and Number Partitioning. 39(3):378–406, 2013.

[47] David S Johnson, Murray Hill, Cecilia R Aragon, Lyle A Mcgeoch, and Catherine Schevon. OPTIMIZATION BY SIMULATED ANNEALING : AN EXPERIMENTAL NUMBER PARTITIONING. 39(3), 1991.

[48] Jana C. Jokerst, Jason M. Emory, and Charles S. Henry. Advances in microfluidics for environmental analysis. *The Analyst*, 137(1):24, 2012.

[49] Andrew B Kahng and Chung-wen Albert Tsao. Planar-DME : A Single-Layer Zero-Skew Clock Tree Router. 15(1):8–19, 1996.

[50] Richard M. Karp. Reducibility Among Combinatorial Problems. *Complexity of Computer Computations*, 43(4):85 – 103, 1972.

[51] Jungkyu Kim, Erik C Jensen, Mischa Megens, Bernhard Boser, and Richard a Mathies. Integrated microfluidic bioprocessor for solid phase capture immunoassays. *Lab on a chip*, 11(18):3106–12, sep 2011.

[52] I Klammer, a Buchenauer, H Fassbender, R Schlierf, G Dura, W Mokwa, and U Schnakenberg. Numerical analysis and characterization of bionic valves for microfluidic PDMS-based systems. *Journal of Micromechanics and Microengineering*, 17:S122–S127, 2007.

[53] Casimir Kuratowski. Sur le problme des courbes gauches en topologie. *Fundamenta Mathematicae*, 15(1):271–283, 1930.

[54] C Y Lee. An Algorithm for Path Connections and Its Applications. *IRE Transactions on Electronic Computers*, EC-10(3):1389–1401, 1961.

[55] Chung-Cheng Lee, Guodong Sui, Arkadij Elizarov, Chengyi Jenny Shu, Young-Shik Shin, Alek N. Dooley, Jiang Huang, Antoine Daridon, Paul Wyatt, David Stout, Hartmuth C. Kolb, Owen N. Witte, Nagichettiar Satyamurthy, James R. Heath, Michael E. Phelps, Stephen R. Quake, and Hsian-Rong Tseng. Multistep synthesis of a radiolabeled imaging

probe using integrated microfluidics. *Science (New York, N.Y.)*, 310(5755):1793–1796, 2005.

[56] Baichen Li, Lin Li, Allan Guan, Quan Dong, Kangcheng Ruan, Ronggui Hu, and Zhenyu Li. A Smartphone Controlled Handheld Microfluidic Liquid Handling System. *Lab on a Chip*, 14(20):4085–92, 2014.

[57] Y. C. Lim, A. Z. Kouzani, and W. Duan. Lab-on-a-chip: a component view. *Microsystem Technologies*, 16(12):1995–2015, sep 2010.

[58] Chun-Xun Lin, Chih-Hung Liu, I-Che Chen, D. T. Lee, and Tsung-Yi Ho. An Efficient Bi-criteria Flow Channel Routing Algorithm For Flow-based Microfluidic Biochips. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference - DAC '14*, pages 1–6, New York, New York, USA, 2014. ACM Press.

[59] Joshua S. Marcus, W. French Anderson, and Stephen R. Quake. Microfluidic single-cell mRNA isolation and analysis. *Analytical chemistry*, 78(9):3084–9, may 2006.

[60] Daniel Mark, Stefan Haeberle, Günter Roth, Felix von Stetten, and Roland Zengerle. Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Chemical Society reviews*, 39(3):1153–82, mar 2010.

[61] Jeffrey McDaniel, Auralila Baez, Brian Crites, Aditya Tammewar, and Philip Brisk. Design and Verification Tools for Continuous Fluid Flow-based Microfluidic Devices. In *Asia and South Pacific Design Automation Conference (ASPDAC)*, 2013.

[62] Jeffrey McDaniel, Christopher Curtis, and Philip Brisk. Automatic Synthesis of Microfluidic Large Scale Integration Chips from a Domain-Specific Language. In *Biological Circuits and Systems (BioCAS)*, number 1, pages 3–6. Ieee, oct 2013.

[63] Jeffrey McDaniel, Daniel T Grissom, and Philip Brisk. Multi-terminal PCB Escape Routing for Digital Microfluidic Biochips using Negotiated Congestion. In *22nd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, number c, 2014.

[64] Larry McMurchie and Carl Ebeling. PathFinder : A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the Third International ACM Symposium on Field-Programmable Gate Arrays (FPGA '95)*, pages 111–117, 1995.

[65] Jessica Melin and Stephen R. Quake. Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annual review of biophysics and biomolecular structure*, 36:213–231, jan 2007.

[66] Wajid Hassan Minhass, Paul Pop, and Jan Madsen. System-Level Modeling and Synthesis of Flow-Based Microfluidic Biochips. In *CASES*, pages 225–233, 2011.

[67] Wajid Hassan Minhass, Paul Pop, and Jan Madsen. Synthesis of Biochemical Applications on Flow-Based Microfluidic Biochips using Constraint Programming. In *DTIP*, number April, 2012.

[68] Wajid Hassan Minhass, Paul Pop, Jan Madsen, and Felician Stefan Blaga. Architectural Synthesis of Flow-Based Microfluidic Large-Scale Integration Biochips. In *CASES 2012*, pages 181–190, 2012.

[69] Kwang W Oh, Kangsun Lee, Byungwook Ahn, and Edward P Furlani. Design of pressure-driven microfluidic networks using electric circuit analogy. *Lab on a Chip*, 12(3):515–545, feb 2012.

[70] R Pal, M Yang, R Lin, B N Johnson, N Srivastava, S Z Razzacki, K J Chomistek, D C Heldsinger, R M Haque, V M Ugaz, P K Thwar, Z Chen, K Alfano, M B Yim, M Krishnan, A O Fuller, R G Larson, D T Burke, and M A Burns. An integrated microfluidic device for influenza and other genetic analyses. *Lab on a chip*, 5(10):1024–32, oct 2005.

[71] Jefferey Perkel. Microfluidics: bringing new things to life science. *Life Science Technologies*, AAAS/Scien:975–977, 2008.

[72] Michael Raagaard. Placement Algorithm for Flow-Based Microfluidic Biochips. 2014.

[73] Minsoung Rhee and Mark a Burns. Microfluidic assembly blocks. *Lab on a chip*, 8(8):1365–73, 2008.

[74] Eric K Sackmann, Anna L Fulton, and David J Beebe. The present and future role of microfluidics in biomedical research. *Nature*, 507(7491):181–9, 2014.

[75] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice*. WSPC, 1 edition, 1999.

[76] Carl Sechen. *VLSI Placement and Global Routing Using Simulated Annealing*. Springer Science & Business Media, 2012.

[77] Jonathan Siegrist, Mary Amasia, Navdeep Singh, Debjyoti Banerjee, and Marc Madou. Numerical modeling and experimental validation of uniform microchamber filling in centrifugal microfluidics. *Lab on a chip*, 10(C):876–886, 2010.

[78] Larisa N Soldatova, Wayne Aubrey, Ross D King, and Amanda Clare. The EXACT description of biomedical protocols. *Bioinformatics (Oxford, England)*, 24(13):i295–303, jul 2008.

[79] Bill Thies and Nada Amin. Programmable Microfluidics. *October*, pages 191–193, 2007.

[80] Todd Thorsen, Sebastian J Maerkl, and Stephen R. Quake. Microfluidic large-scale integration. *Science (New York, N.Y.)*, 298(5593):580–584, 2002.

[81] Kai-han Tseng, Sheng-chi You, and Tsung-yi Ho. A Network-Flow Based Valve-Switching Aware Binding Algorithm for Flow-Based Microfluidic Biochips. pages 213–218, 2013.

[82] Kai-Han Tseng, Sheng-Chi You, Jhe-Yu Liou, and Tsung-Yi Ho. A Top-Down Synthesis Methodology for Flow-Based Microfluidic Biochips Considering Valve-Switching Minimization. In *ISPD*, pages 123–129, 2013.

[83] Marc Alexander Unger, Hou-Pu Chou, Todd Thorsen, Axel Scherer, and Stephen R. Quake. Monolithic Microfabricated Valves and Pumps by Multilayer Soft Lithography. *Science*, 288(5463):113–116, apr 2000.

[84] John Paul Urbanski, William Thies, Christopher Rhodes, Saman Amarasinghe, and Todd Thorsen. Digital microfluidics using soft lithography. *Lab on a chip*, 6(1):96–104, jan 2006.

[85] Kristofer Vorwerk, Andrew Kennings, Jonathan Greene, and Doris T Chen. Improving Annealing Via Directed Moves. pages 363–370, 2007.

[86] George M Whitesides. The origins and the future of microfluidics. *Nature*, 442(7101):368–73, 2006.

[87] Martin D.F. Wong, Hua Xiang, and Xiaoping Tang. Min-cost flow-based algorithm for simultaneous pin assignment and routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7):870–878, jul 2003.

[88] Angela R Wu, Joseph B Hiatt, Rong Lu, Joanne L Attema, Neethan a Lobo, Irving L Weissman, Michael F Clarke, and Stephen R. Quake. Automated microfluidic chromatin immunoprecipitation from 2,000 cells. *Lab on a chip*, 9(10):1365–70, may 2009.

[89] Angela R Wu, Tiara L A Kawahara, Nicole A Rapicavoli, Jan van Riggelen, Emelyn H Shroff, Liwen Xu, Dean W Felsher, Howard Y Chang, and Stephen R. Quake. High throughput automated chromatin immunoprecipitation as a platform for drug screening and antibody validation. *Lab on a chip*, 12(12):2190–8, jun 2012.

[90] Paul Yager, Thayne Edwards, Elain Fu, Kristen Helton, Kjell Nelson, Milton R Tam, and Bernhard H Weigl. Microfluidic diagnostic technologies for global public health. *Nature*, 442(7101):412–8, jul 2006.

[91] Tan Yan and Martin D.F. Wong. BSG-Route: A length-matching router for general topology. *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 499–505, nov 2008.

[92] Hailong Yao, Tsung-Yi Ho, and Yici Cai. PACOR : Practical Control-Layer Routing Flow with Length-Matching Constraint for Flow-Based Microfluidic Biochips. In *Design Automation Conference*, 2015.

[93] Jeong Yeol Yoon and Bumsang Kim. Lab-on-a-chip pathogen sensors for food safety. *Sensors (Switzerland)*, 12(8):10713–10741, 2012.

Appendix A

Sequencing Graphs

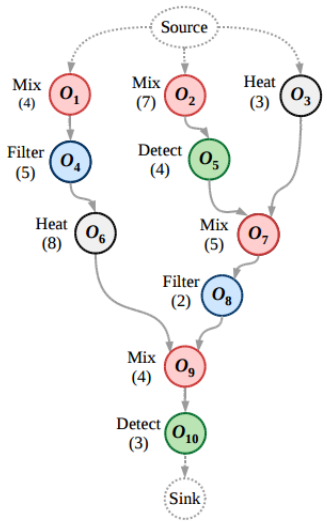


Figure A.1: Sequencing graph for synthetic benchmark 1 from Denmark Technical University (DTU) [2]. The benchmark has 10 operations.

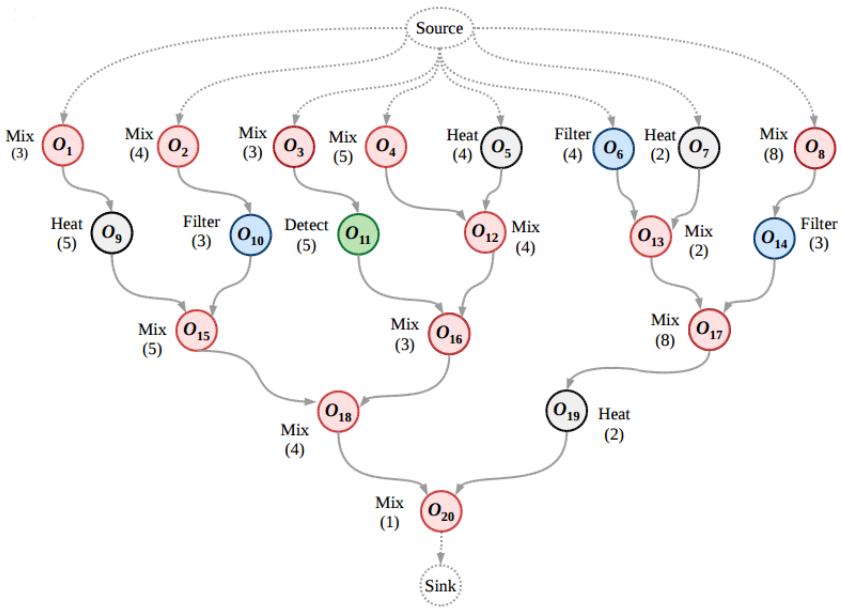


Figure A.2: Sequencing graph for synthetic benchmark 2 from Denmark Technical University (DTU) [2]. The benchmark has 20 operations.

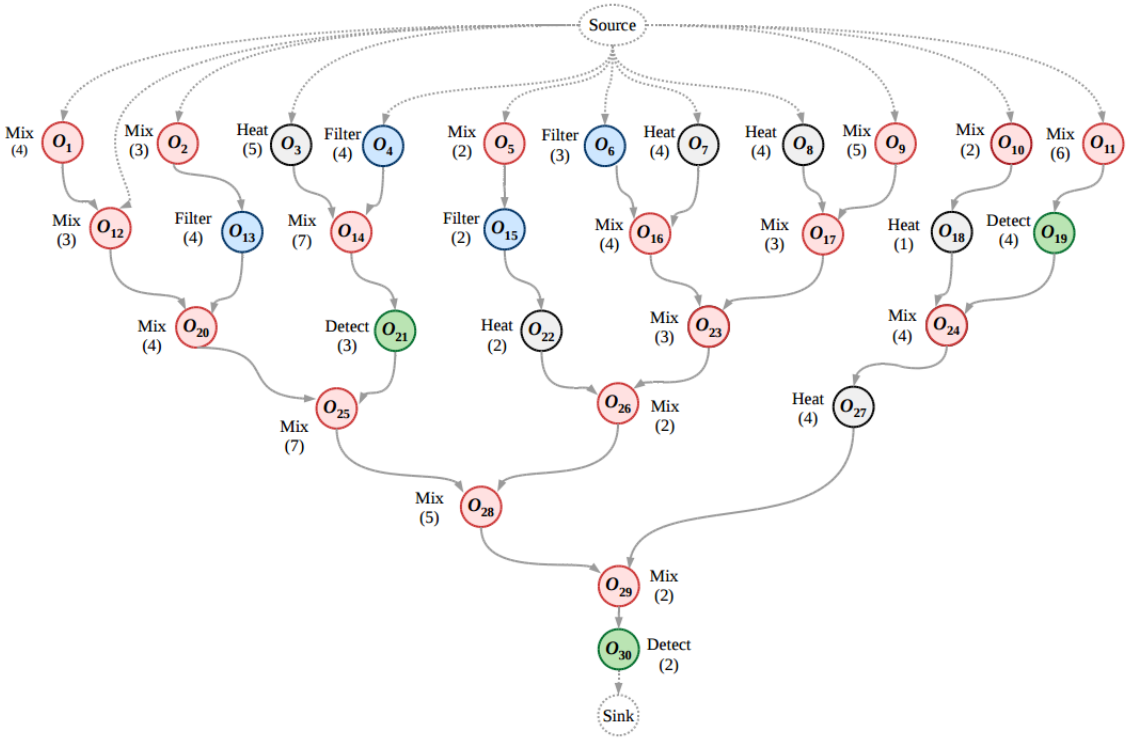


Figure A.3: Sequencing graph for synthetic benchmark 3 from Denmark Technical University (DTU) [2]. The benchmark has 30 operations.

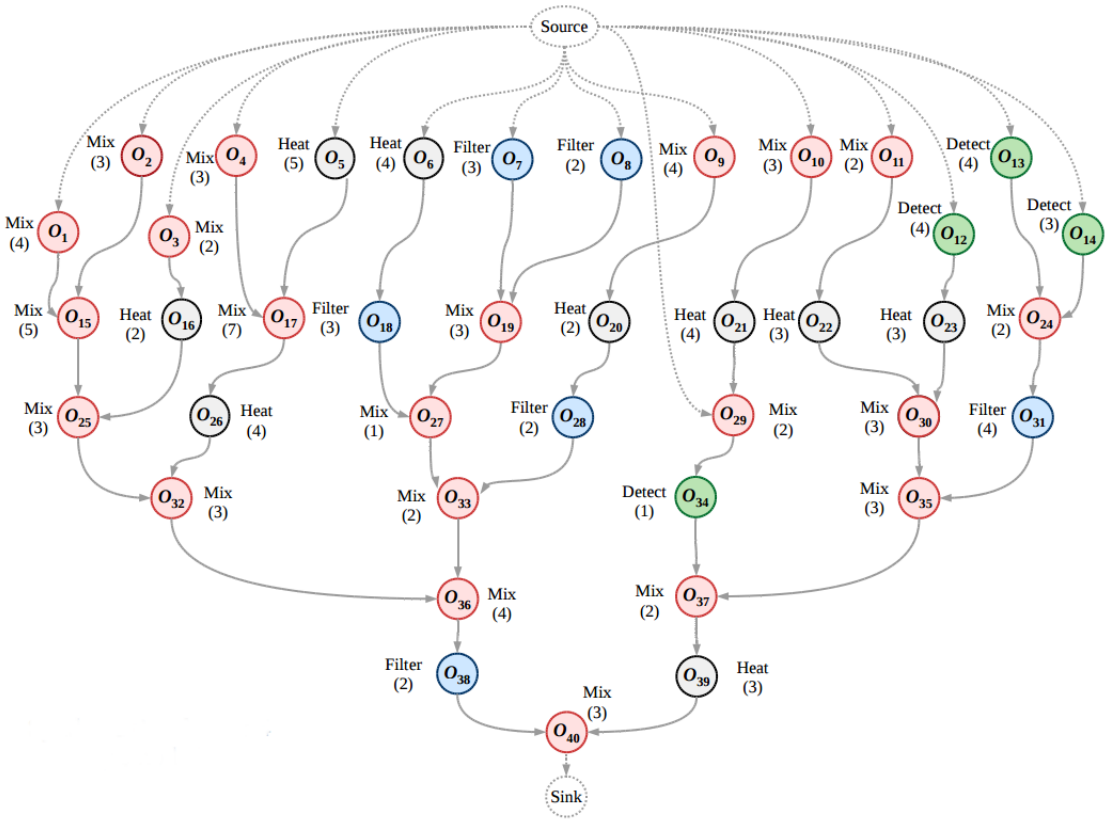


Figure A.4: Sequencing graph for synthetic benchmark 4 from Denmark Technical University (DTU) [2]. The benchmark has 40 operations.

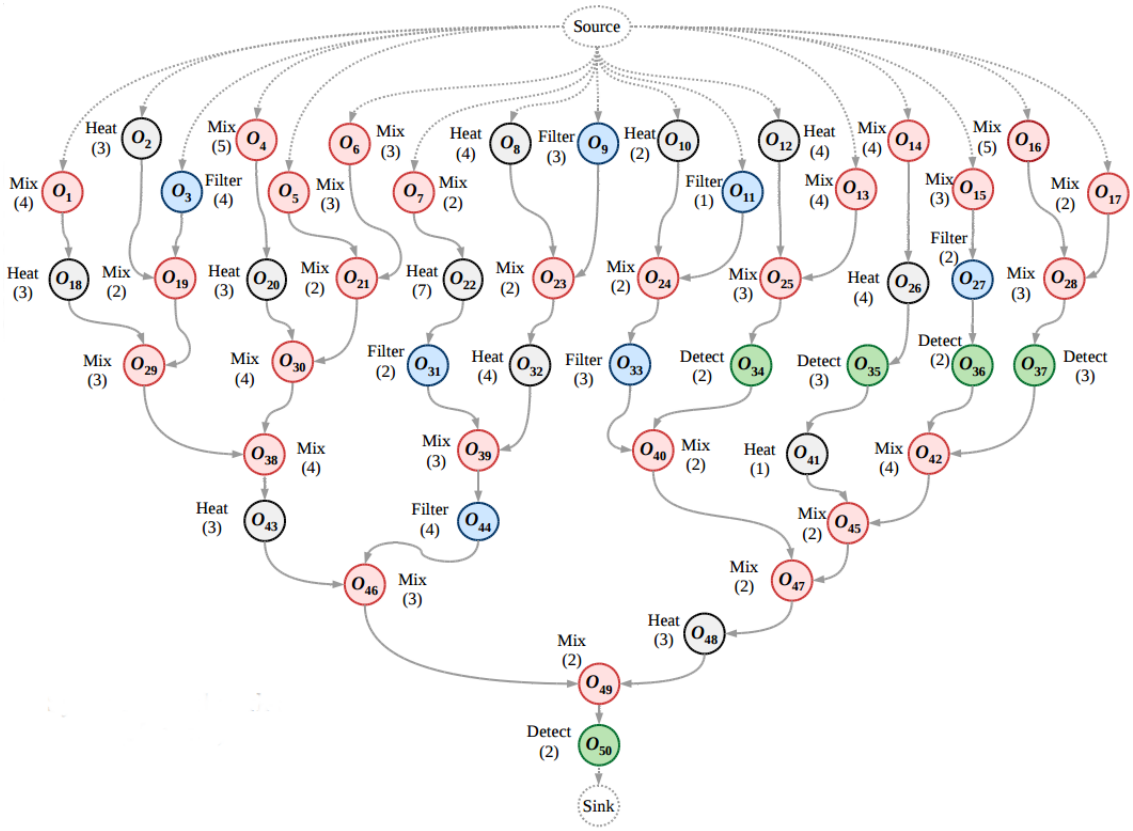


Figure A.5: Sequencing graph for synthetic benchmark 5 from Denmark Technical University (DTU) [2]. The benchmark has 50 operations.

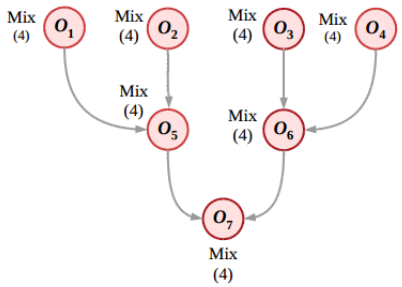


Figure A.6: Sequencing graph for polymerase chain reaction (PCR) benchmark from Denmark Technical University (DTU) [2]. The benchmark has 7 operations.

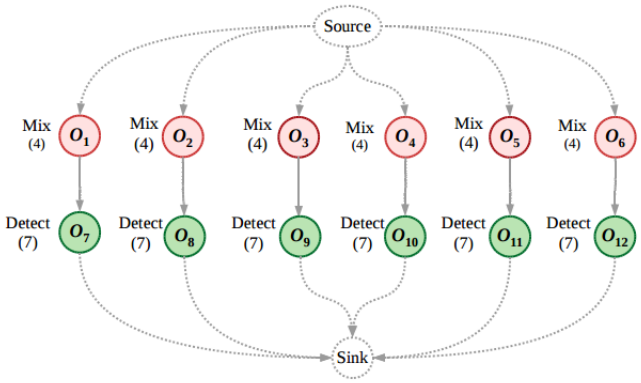


Figure A.7: Sequencing graph for in-vitro diagnostics (IVD) benchmark from Denmark Technical University (DTU) [2]. The benchmark has 22 operations.

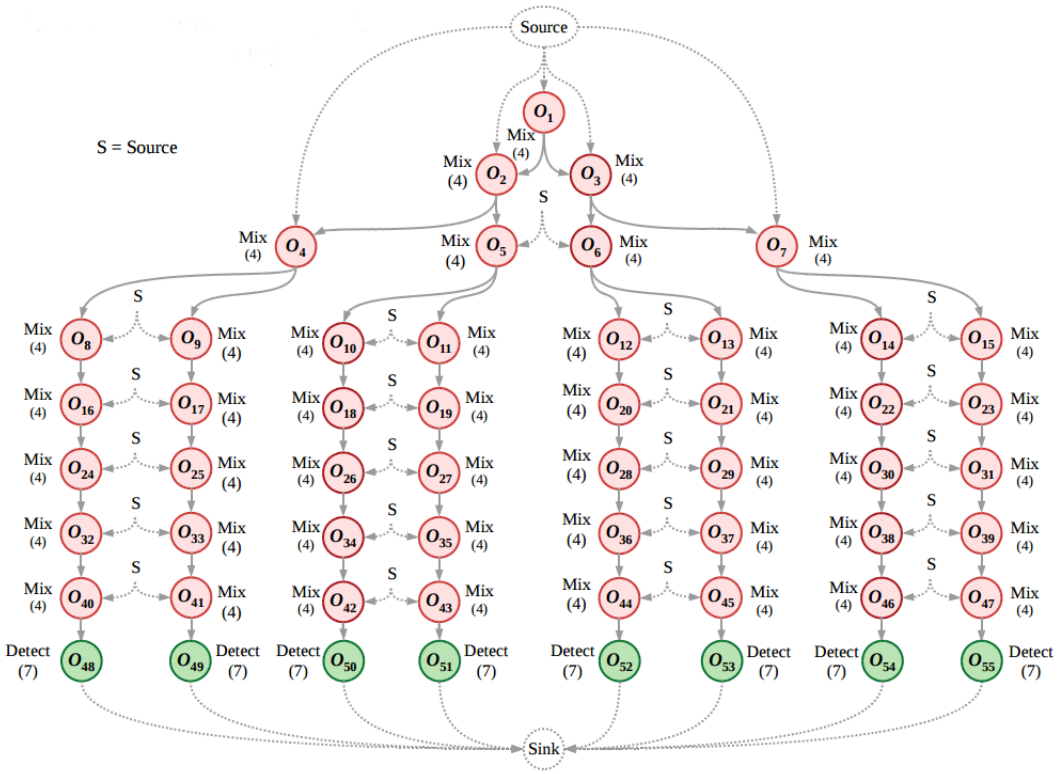


Figure A.8: Sequencing graph for colorimetric protein assay (CPA) benchmark from Denmark Technical University (DTU) [2]. The benchmark has 55 operations.