2016

Fluigi: an end-to-end software workflow for microfluidic design

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

FLUIGI: AN END-TO-END SOFTWARE WORKFLOW FOR MICROFLUIDIC DESIGN

by

HAIYAO HUANG

M.Eng., Massachusetts Institute of Technology, 2007

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2016

Approved by

First Reader

Douglas Densmore, PhD
Associate Professor of Electrical and Computer Engineering

Second Reader

Ayse Coskun, PhD
Associate Professor of Electrical and Computer Engineering

Third Reader

Ahmad S. Khalil, PhD
Assistant Professor of Biomedical Engineering

Fourth Reader

Martin Herbordt, PhD
Professor of Electrical and Computer Engineering

Acknowledgments

I would like thank my advisor Douglas Densmore for giving me the chance to continue working in synthetic biology after I had a minor crisis of conscience working in industry. The last four years have been a wonderful experience.

I would like to thank Professor Ahmad Khalil for letting me borrow his lab space and equipment for making and testing microfluidics, and for the initial ideas of exploring the space of automating microfluidic design; and his graduate students Ali Beyzavi and Brandon Wong for teaching me how to fabricate molds in the clean room.

I'd like to thank Professor Ayse Coskun and Professor Martin Herbordt for being on my committee and providing valuable feedback during the prospectus.

I'd like to thank Swapnil Bhatia for his assistance with the early ideas on using biological circuits in microfluidics and Ernst Oberortner for his assistance working with `ANTLR`. I'd also like to thank Traci Haddock for keeping peace in the lab and during lab meetings and preventing the members of the computational side of things from damaging themselves and their experiments in the wet lab.

I'd like to thank my labmates in the CIDAR lab, most notably Ryan Silva, Sonya Iverson, and Aaron Heuckroth, for getting the Fluigi project off the ground, and for some unforgettable memories of white elephant gift swaps and that one trip to Maine.

Finally, I'd like to thank my husband Jim Waldrop for his continued moral support during my time as a graduate student and the household felines for keeping me company while I coded.

FLUIGI: AN END-TO-END SOFTWARE WORKFLOW FOR MICROFLUIDIC DESIGN

HAIYAO HUANG

Boston University, College of Engineering, 2016

Major Professor: Douglas M. Densmore, PhD
Associate Professor of Electrical and Computer Engineering

ABSTRACT

One goal of synthetic biology is to design and build genetic circuits in living cells for a range of applications with implications in health, materials, and sensing. Computational design methodologies allow for increased performance and reliability of these circuits. Major challenges that remain include increasing the scalability and robustness of engineered biological systems and streamlining and automating the synthetic biology workflow of specify-design-build-test.

I summarize the advances in microfluidic technology, particularly microfluidic large scale integration, that can be used to address the challenges facing each step of the synthetic biology workflow for genetic circuits. Microfluidic technologies allow precise control over the flow of biological content within microscale devices, and thus may provide more reliable and scalable construction of synthetic biological systems. However, adoption of microfluidics for synthetic biology has been slow due to the expert knowledge and equipment needed to fabricate and control devices. I present an end-to-end workflow for a computer-aided-design (CAD) tool, Fluigi, for designing microfluidic devices and for integrating biological Boolean genetic circuits with mi-

crofluidics. The workflow starts with a "netlist" input describing the connectivity of microfluidic device to be designed, and proceeds through placement, routing, and design rule checking in a process analogous to electronic computer aided design (CAD). The output is an image of the device for printing as a mask for photolithography or for computer numerical control (CNC) machining. I also introduced a second workflow to allocate biological circuits to microfluidic devices and to generate the valve control scheme to enable biological computation on the device.

I used the CAD workflow to generate 15 designs including gradient generators, rotary pumps, and devices for housing biological circuits. I fabricated two designs, a gradient generator with CNC machining and a device for computing a biological XOR function with multilayer soft lithography, and verified their functions with dye. My efforts here show a first end-to-end demonstration of an extensible and foundational microfluidic CAD tool from design concept to fabricated device. This work provides a platform that when completed will automatically synthesize high level functional and performance specifications into fully realized microfluidic hardware, control software, and synthetic biological wetware.

Contents

List of Tables

List of Figures

List of Abbreviations

BST	.....	Binary Search Tree
CAD	.....	Computer-Aided Design
CNC	.....	Computer Numerical Control
DNA	.....	Deoxyribonucleic Acid
FPGA	.....	Field Programmable Gate Array
GFP	.....	Green Fluorescent Protein
MAGE	.....	Multiplex Automated Genome Engineering
mLSI	.....	Microfluidic Large Scale Integration
mRNA	.....	Messenger Ribonucleic Acid
PDMS	.....	Polydimethylsiloxane
ORF	.....	Open Reading Frame
SBOL	.....	Synthetic Biology Open Language
TLFM	.....	Time-Lapse Fluorescent Microscopy
VLSI	.....	Very Large Scale Integration

Chapter 1

Introduction

Synthetic biology as a field has the potential to offer game-changing breakthroughs in the fields of alternative energy, drug discovery, customized medicine, and alternative computing. The main challenges facing scientists, particularly those working in micro-organisms such as *E. coli* or yeast, are scaling up the number of experiments and the ability to quickly and accurately reproduce previous experiments. Microfluidics, particularly continuous flow based systems and microfluidic large scale integration, provides a technology platform for chemical and biological experiments that can greatly benefit the synthetic biology community. Specific details and examples of using microfluidics in the synthetic biology workflow of specify-design-assemble-verify are described in Chapter 2.

Currently, the majority of synthetic biology labs do not adopt microfluidics for the simple reason that microfluidics are hard to both design and manufacture. Few synthetic biology labs have personnel with detailed knowledge of fluid dynamics. The process to design a microfluidic device involves drawing every device feature and channel by hand in a graphics program such as Adobe Illustrator or AutoCAD. The designer also has to keep track of and adjust all the required spacing between elements and between layers of the device. The design process can take from several days for a simple device to several weeks for a complex device. Small changes such as altering the size of elements or spacing between elements may mean redrawing large sections of the design to account for the overall design requirements. For microfluidic

large scale integration, devices can contain thousands of elements, making layout by hand increasingly time-consuming and error-prone. In addition, the equipment and cleanroom space needed for device manufacturing is beyond the purview for many labs. With the current manufacturing process of multi-layer soft lithography, it takes approximately 7-10 days to make a device once a design has been finalized and a photomask printed. Repeated iterations of device designs would take weeks or months with current technologies, making it vital that the device designs are error-free before fabrication.

The goal now is to reduce the difficulty of designing and fabricating microfluidic devices so more synthetic biology labs will take advantage of the technology for their experiments. This thesis focuses on removing some of the barriers to microfluidic design by introducing a CAD tool for an end-to-end workflow from a textual description of a microfluidic device to the generation of a photomask. The scope of the project is limited to continous flow based systems as those systems are the most amenable to the cell growth and monitoring experiments commonly used in synthetic biology. The workflow captures and formalizes the design parameters that would otherwise be derived from trial and error and allows lab-specific design choices to be easily shared in the form of initialization files. This ensures that expert knowledge is retained and reduces the learning curve of new designers.

The end-to-end workflow begins with a new netlist format for describing the features on microfluidic devices and the connections between the features. The netlist format contains commonly used microfluidic design elements such as ports, mixers, cell chambers, and multiplexers, and can be easily extended to describe additional features. The full specification for the netlist format is in Chapter 4.

The netlist is converted to a graph representation of the microfluidic device, and the device undergoes automated layout and design rule checking. Algorithms for

placement and routing from electronic design automation were adapted to work with the design constraints of microfluidic devices. Simulated annealing is used for placement, and Hadlock's variation of maze routing is used for routing. Design rule checking is performed with a search of intersections between device features that reduces to a 1-D interval search. These processes are described in detail in Chapter 5. The result of this is a vector graphics file of the design for manufacturing.

This workflow was used to build and test a prototype device for solving one of the challenges facing synthetic biology: the lack of non-interferring genetic parts in large genetic networks. Microfluidics could allow reuse of existing genetic parts by separating the parts both spatially and temporally. Previous attempts at solving this challenge are described in Chapter 3. A four chamber device that allowed fluid routing between any two chambers was designed with this workflow. The photomask generated by this workflow was used to fabricate the device through multilayer soft lithography, and the device was tested with dye. An alternative method of fabrication with a desktop computer numerical controlled (CNC) mill was also explored. Where it would take a week to fabricated a device with multilayer soft lithography, a CNC mill can be used to fabricated devices in less than an hour. The workflow was used to generate the design for a gradient generator, and the final device was fabricated with the CNC mill and tested with dye. The results of these two experiments are described in Chapter 6.

The contributions of this work to the fields of synthetic biology, microfluidics, and design automation are two-fold. First, I provide the first end-to-end workflow for microfluidic design automation that includes a new netlist format for describing microfluidic devices, automated layout through place-and-route, and design rule checking. Second, I provide a prototype device built with the workflow for directing communications between different cell populations that may increase the scalability

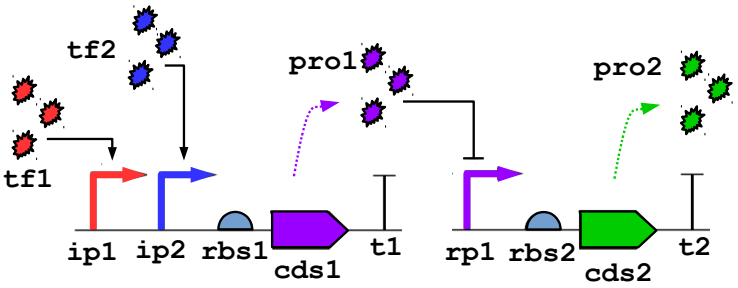
of biological computation. This work has to potential to unlock a paradigm shift in both microfluidic design and synthetic biology by harnessing the capabilities of design automation.

Chapter 2

Background

(This chapter was originally published as Huang, H. and Densmore, D. (2014). Integration of microfluidics into the synthetic biology design flow. Lab Chip, 14:3459 - 3474. (Huang and Densmore, 2014b).)

Over the last decade, synthetic biology has emerged as a field with potential applications in diverse fields including pharmaceuticals, biofuels, and materials. The engineering of a microbial production pathway for artemisinic acid (a precursor for antimalarial drugs) (Ro et al., 2006) has lowered the production costs of those drugs from $2.40 to $0.40 per dose, expanding the number of patients who can afford the treatment for a disease that kills millions (Densmore and Hassoun, 2012). Several companies from around the world (Gevo in Englewood, Colorado, Butamax in Wilmington, Delaware, and Butalco in Fuerigen, Switzerland) have all developed methods to increase the yield of isobutanol or butanol for commercial biofuels using various strains of yeast (Peralta-Yahya et al., 2012), while Amyris has adapted the mevalonate and deoxyxyulose phosphate metabolic pathways in yeast to ferment farnesene (Westfall and Gardner, 2011). Engineered bacteriophages (viruses targeting bacteria) have been used to destroy biofilms (Lu and Collins, 2007) and resensitize otherwise antibiotic-resistant strains of bacteria (Lu and Collins, 2009), which could extend the effectiveness of current drug therapies in the face of rising antibiotic resistance. However, despite these success stories, the field of synthetic biology faces challenges in workflow acceleration and automation as it seeks to scale from single prototypes



tf1	tf2	pro1	pro2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Figure 2·1: A genetic NOR gate, and accompanying truth table. The presence of transcription factors *tf1* or *tf2* activate the inducible promoters *ip1* and *ip2* respectively and allow for production of *pro1*, coded for by *cds1*. The protein *pro1* acts as a transcription factor on repressible promoter *rp1* and blocks production of *pro2*, coded for by *cds1*. The final output, *pro2* is only produced when both transcriptional factors *tf1* or *tf2* are absent.

to commercial enterprises. The field of microfluidics has proved successful in a wide range of applications in biology and has the potential to address these challenges.

2.1 Engineering biology

Synthetic biology seeks to manipulate the structure and functions of deoxyribonucleic acid (DNA) to create new biological systems according to engineering principles. The primary paradigm in the field is to identify biological primitives involved in the transformation of DNA to a protein and encapsulate these individual DNA sequences as "parts"(Voigt, 2006). Synthetic biology has taken advantage of the modularity in the structure of bacterial open reading frames (ORFs) to abstract portions of the sequence as "parts". These "parts" include sequences for promoters, ribosome binding sites, coding regions, and terminators, and can be combined to create functional "devices" which can be introduced into living organisms such as bacteria, yeast, or mammalian cells (Canton et al., 2008; Khalil et al., 2012; Carr and Church, 2009). As logical frameworks such as repressible systems exhibiting the behavior of inverters already exist in gene expression systems, synthetic biology seeks to harness and re-

engineer these natural logic systems for other applications.

A genetic device that implements the Boolean NOR function and the corresponding truth table are shown in Figure 2·1. This device is designed to function in prokaryotic systems. In this circuit, the inputs are the transcription factors *tf1* and *tf2*, which control the inducible promoters *ip1* and *ip2* such that transcription only takes place when one or both of the transcription factors are present. Transcription continues until terminator *t1*, and the resulting messenger RNA (mRNA) is translated into the protein *pro1* (coded for by the coding region *cds1*). The protein *pro1* acts as a transcription factor for the repressible promoter *rp1*, preventing transcription starting at *rp1* if it is present. The output of this system is the protein *pro2* (coded for by the coding region *cds2*), which is only expressed if neither *tf1* or *tf2* are present. In the absence of *tf1* and *tf2*, *pro1* is not expressed. Transcription starts at *rp1* and continues through *t2*, allowing *pro2* to be expressed. This behavior of this device emulates that of the Boolean NOR function, in which the output is true if and only if both inputs are false. NOR and NAND functions are functionally complete and can be used to build all other Boolean logic functions, thus allowing these devices to be the basis for biological computation. As such, many different implementations of these functions exist in synthetic biology (Goñi-Moreno and Amos, 2012; Wang et al., 2011; Bonnet et al., 2013).

While many different classes of biological devices such as oscillators (Purcell et al., 2010), filters (Sohka et al., 2009), noise generators (Lu et al., 2008), and the beginnings of analog computation (Daniel et al., 2013) exist, I focus here instead on the biological devices implementing Boolean logic (Brophy and Voigt, 2014) and other extensions of the digital abstraction found in electronics. I use the digital abstraction not to replicate silicon-based computing, but as a method for designing robust biological circuits that are insensitive to noise and can be tuned to specific input conditions.

Additionally, the digital abstraction is well understood, and numerous techniques have been developed for its description, synthesis, and verification. The introduction of Boolean logic and memory devices into biological systems leads to new applications and potential methods of computation for solving otherwise computationally intensive and complex problems (Haynes et al., 2008; Baumgardner et al., 2009).

2.1.1 Boolean logic in biological devices

Biological logic devices can be used to detect specific combinations of chemical or environmental triggers for targeted pharmaceutical and biotechnology applications (Tamsir et al., 2011; Gupta et al., 2013). One application for biological logic circuits is in the field of cancer research (Ruder et al., 2011; Shankar and Pillai, 2011), where the use of digital logic provides the necessary specificity for targeting strains of cancer cells while leaving other cells unharmed (Anderson et al., 2006; Xie et al., 2011). Nissim et al. (Nissim and Bar-Ziv, 2010) introduce a tunable dual promoter system that implements the Boolean function AND to target cancer cells while ignoring premalignant cells.

While devices implementing two-input Boolean logic functions are useful in synthetic biology, more complex computation would allow for applications such as the biological sensing of multiple chemical species in the same device, the identification of specific genetic markers, and environmentally tailored drug dosage responses (Purnick and Weiss, 2009). One way of constructing more complex functions is to increase the layers of logic in the genetic device. This method was used by Moon et al. to create a four input transcriptional AND gate (Moon et al., 2012) with eleven orthogonal (non-interfering) regulatory proteins made from two layers of two-input AND gates. Another tactic is to separate the larger function into smaller functions and place devices implementing the smaller functions into different cells. These cells then communicate with each other through intercellular signaling pathways. All six-

teen two-input functions have been built from *E. coli* cells containing NOR gates that communicate through the quorum sensing pathway (Tamsir et al., 2011). Distributed computing (Macía et al., 2012) has also been implemented in yeast with the development of both a 2-to-1 multiplexer and a basic addition circuit (Regot et al., 2011).

2.1.2 Memory and state in biological devices

The next step in increasing complexity of computation is to generate the concept of memory or "state", such that the cell remembers what has previously happened and takes that into account in new calculations. One approach is to use recombinases, enzymes used by bacteriophages to manipulate their host's genome, to turn specific DNA sequences on and off by switching the orientation of the DNA (Khalil and Collins, 2010). Memory devices and counters have been integrated into cells through the use of recombinase-based circuits (Ham et al., 2008; Friedland et al., 2009). The use of recombinase has also provided synthetic biologists with a form of rewrite-able and addressable data storage capable of information storage through over 100 cell divisions and through repeated switching without losing performance (Bonnet et al., 2012). More recently, Siuti et al. have used recombinase-based circuits to implement all 16 two-input Boolean logic functions with stable DNA-encoded memory of events in *E. coli* without requiring cascades of multiple gates (Siuti et al., 2013).

2.1.3 Specify-Design-Assemble-Verify workflow

The practice of synthetic biology typically follows an iterative process of specification, design, assembly, and verification. The process begins with the specification of the function of the novel genetic device either by hand or with one of the new description languages such as Eugene (Bilitchenko et al., 2011a; Bilitchenko et al., 2011b), GEC (Pedersen and Phillips, 2009), or Proto (Beal et al., 2011). In the design phase, bio-

logical parts are selected from repositories to implement the specified function. Tools such as GenoCAD (Cai et al., 2010), j5 (Hillson et al., 2012), or Clotho (Xia et al., 2011) may aid the design process. Assembly of a novel genetic device starts with obtaining the parts of interest either by isolating segments of DNA from natural sources or by *de novo* synthesis through companies such as DNA2.0, GeneArt (Densmore and Hassoun, 2012), or Gen9 (Goldberg, 2013). Parts are assembled into devices by joining the segments of DNA together using restriction enzymes (proteins that cut DNA at certain sequences) and ligases (proteins that create new bonds between DNA bases). Common assembly techniques, including BioBrick (Shetty et al., 2008), BglBrick (Anderson et al., 2010), Gibson (Gibson et al., 2009), GoldenGate (Engler et al., 2008), the Modular Overlap-Directed Assembly with Linkers (MODAL) (Casini et al., 2014), and modular cloning (MoClo) (Weber et al., 2011), are reviewed in detail elsewhere (Ellis et al., 2011). Software tools are being developed to automate and optimize the assembly process (Densmore et al., 2010; Appleton et al., 2014).

The completed device is then inserted into a host organism, commonly *E. coli* for prokaryotic systems and yeast (*Saccharomyces cerevisiae*, a model organism for eukaryotic studies) for eukaryotic systems, and the organism is grown under a variety of conditions to test the function of the device as compared to the original specifications. For verification purposes, fluorescent proteins are often used as a substitute for the gene of interest as their expression is more easily measured through flow cytometry to determine the efficiency of the device under test. The fluorescent protein is replaced by the protein of interest in the final application. At this stage, the genetic device may be further refined based on the gathered data, or the host cells may be grown for harvesting more copies of the device.

As new techniques in synthetic biology generate large libraries of thousands of part variations and combinatoric devices, increased throughput and automation are needed

to test and characterize these constructs to allow the acceleration of the specify-design-assemble-verify development cycle for synthetic biology. To lower the barrier of entry into the field, synthetic biology needs to take advantage of the improvement in automation and computer-aided design tools. Software tools for synthetic biology have grown in the past years to encompass a variety from rules-based constraint languages to gene designers to basic simulators. Standards such as the Synthetic Biology Open Language (SBOL) (Galdzicki et al., 2012) are being developed to facilitate information exchange between various tools in the toolchain and will be vital in building end-to-end workflows (Beal et al., 2012) in synthetic biology from high level languages to compilation from the language to biological parts to DNA assembly. While the available tools are still lagging behind the current state of the art of biological research, the development of these tools helps with refining the rules of genetic design for future applications (Lux et al., 2012).

2.2 Microfluidics

Microfluidics is comprised of the analytical systems and tools for the study and manipulation of small volumes of liquids, typically at micro and nano liter scales. The advantages of microfluidics come from the decrease in scale, which allows for more predictable fluid flow, decreasing the amount of reagents needed for reactions, and smaller devices and experiment setups. Microfluidics have been used for chemical analysis (Whitesides, 2006) and PCR (Zhang et al., 2006) as well as a variety of applications in molecular biology (Hamon and Hong, 2013), systems biology (Breslauer et al., 2006), stem cell studies (Gupta et al., 2010; Zhang and Austin, 2012), tissue engineering (Inamdar and Borenstein, 2011), point-of-care diagnostics (Lei, 2012), pathogen detection (Foudeh et al., 2012), and systematic toxicity studies (Sung and Shuler, 2010).

Recently, synthetic biologists have been developing and using microfluidics to study synthetic gene networks and network dynamics (Bennett and Hasty, 2009). Due to the potential in microfluidic systems for the precise control over input stimuli (Wang et al., 2012; Dertinger et al., 2001) and the ability to track single cells (Ferry et al., 2011), there has been an increased interest in microfluidic platforms to further synthetic biology (Lin and Levchenko, 2012). I present the list of challenges facing synthetic biology as the field matures and describe how microfluidics could be used to find solutions to those challenges as well as potential future applications for systems and workflows integrating both synthetic biology and microfluidics.

The field of microfluidics covers a wide range of technologies such as lateral flow tests, linear actuated devices, pressure driven laminar flow, microfluidic large scale integration, segmented flow microfluidics, centrifugal microfluidics, electrokinetics, electrowetting, surface acoustic waves, and dedicated systems for massively parallel analysis (Haeberle and Zengerle, 2007; Mark et al., 2010). For the purposes of this project, I will focus on the subset of microfluidics that offer the most relevance to current problems in the synthetic biology workflow, particularly in spatial and temporal gradient generation, microfluidic large scale integration (mLSI) high-throughput screening, DNA synthesis, and cell culture.

2.2.1 Device physics

In pressure-driven systems, the reduced scale of microfluidic devices results in deterministic fluid flow. The low Reynolds number (a comparison of the forces acting on the flow) of the flow means that the nonlinear and chaotic effects due to turbulence caused by inertial forces are removed, and the flow is restricted to the laminar region. In this flow regime, the behavior of the fluids can be predicted by the size of the fluid channel and the viscosity of the fluid in a manner analogous to Ohm's Law in electrical engineering (Oh et al., 2012), making it easier to simulate and verify the

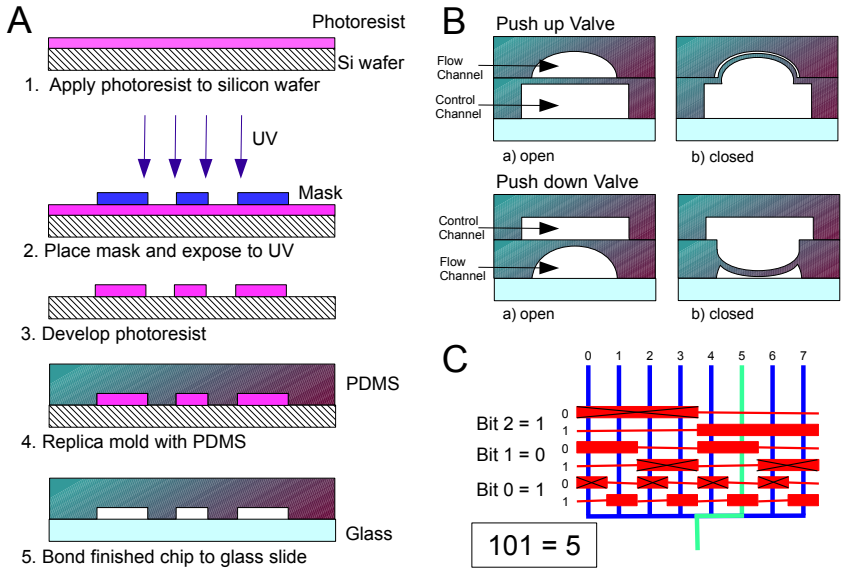


Figure 2·2: *(A)* Fabrication process for soft lithography (adapted from (Zhang and Austin, 2012)). *(B)* Two possible structures (push-up or push-down) for valves in multilayer soft lithography (adapted from (Melin and Quake, 2007)). *(C)* Multiplexer for selection of 8 possible fluid lines, using 6 control lines to represent the 3-bit binary number of the selected line (adapted from Ref. (Thorsen et al., 2002)).

function of the device. The mixing of parallel flows is dominated by diffusion instead of convection, so that the flows only interact at their boundary. This effect can be exploited to generate spatial gradients of chemicals of interest (Dertinger et al., 2001). Mixers (Chou et al., 2001; Stroock et al., 2002) can be used to speed up integration of flows. A detailed synopsis of the physics and fluid mechanics specific to microfluidic devices may be found elsewhere (Squires and Quake, 2005; Kirby, 2010).

2.2.2 Design and fabrication

A widely used fabrication material for microfluidic devices is polydimethylsiloxane (PDMS). The properties of PDMS make it well suited for use in biological applications as it offers flexibility for fabrication, scalability, and potential for long-term growth and monitoring of cells (Balaban et al., 2004). PDMS is optically transparent, chemically inert, impermeable to water but permeable to gases, and non-toxic

to cells (Sia and Whitesides, 2003). The cost of the raw material is around \$0.05 per cm^3 (Unger et al., 2000; Sollier et al., 2011), making it suitable for rapid prototyping and quick design iteration. PDMS-based microfluidics have been used for a variety of purposes in recent years, including as an alternative platform for computation (Thies et al., 2008). Development of the microchemostat (Balagaddé et al., 2005) has allowed for cells to be grown in microfluidic chips for long periods of time, thus allowing for more complex, long-term experiments. However, the low elastic modulus of PDMS makes it unsuitable for high pressure applications as high pressure causes channel deformation in PDMS-based devices (Sollier et al., 2011).

Microfluidic chips are fabricated from PDMS through soft lithography. As many detailed reviews of the process exist (Duffy et al., 1998; Sia and Whitesides, 2003; Weibel et al., 2007), I will only provide a brief description. Photoresist (typically SU-8) is spun out over a substrate of silicon, and a transparency with the chip design is placed over it as a mask. The sandwich of mask, photoresist, and substrate is then exposed to UV light. The mask is removed, and the photoresist washed in developing agent to obtain the master mold. PDMS layers are cast from the master through replica molding. The channels are then sealed against a substrate suitable for imaging and connected to input and control structures. A summary of the process is shown in Figure 2·2A. The entire fabrication process, from the creation of the photomask to the molding of the chip, takes no more than a few days including the turnaround time for printing the photomask. This process does require the experimenter to have access to a high quality cleanroom and purchase specialized fabrication equipment for soft lithography (Elveflow, 2015), or else contract out the fabrication process to a dedicated microfluidic foundry.

2.2.3 Microfluidic large scale integration

An extension of soft lithography, multilayer soft lithography, allows devices to be built of multiple layers of PDMS, typically with one layer as a fluid flow layer and another layer as a control layer with channels pressurized by external actuators (Unger et al., 2000). Fluid flow is controlled by strategic placement of valves in the control layer, which restrict fluid flow when pressurized by causing the PDMS to deform and create a seal across the channel to impede fluid flow (Amin et al., 2009). Two types of valves described by Melin and Quake (Melin and Quake, 2007), push up and push down, are shown in Figure 2·2B. For work with cells, push down valves are preferred as they allow for easier cleaning of the flow channels and chip reuse (Cheong et al., 2009b). Multiple valves may be be controlled by the same pressurized control line, and the optimization problem lies in minimizing the number of control lines needed to operate a chip.

The interaction of the control and flow layer through valves form the basic building block of microfluidic large scale integration (mLSI). As devices made from multilayer soft lithography grow more complicated, an increased number of external pressure lines are needed to control fluid flow. Microfluidic multiplexers, developed by Quake and colleagues (Thorsen et al., 2002), contain combinatorial arrays of binary valve patterns and allow increased fluid manipulation with a minimal number of control inputs such that only $2 \log_2 n$ control lines are needed to access the valves to select from one of n fluid channels. This makes them very suitable for high-throughput applications that require manipulation of hundreds or thousands of fluid elements. The multiplexer shown in Figure 2·2 uses 6 control lines to represent the 3-bit binary number for selecting the fluid channel. Recent advancements in mLSI architecture focus on increasing the number of control elements on the chip through component miniaturization or additional layers, decreasing the reliance on external pneumatic

lines through on-chip logic, and increasing reusability through programmable chips (Araci and Brisk, 2014). Work has also been done on reducing contamination and back flow through the use of a microfluidic serial digital-to-analog pressure converter (Yu et al., 2013).

While CAD and automation have been primarily focused on droplet based digital microfluidics (Su and Chakrabarty, 2005; Su et al., 2006; Chakrabarty and Zeng, 2005; Chakrabarty, 2010) rather than mLSI, a new subset of tools are being developed for layout and optimization of mLSI devices. Earlier tools for mLSI include Biostream, a tool for designing GUIs and control valves for multilayer devices (Thies et al., 2008; Urbanski et al., 2006) (freely available at (Thies et al., 2009)) and Micado (Amin et al., 2009), which automate control valve placement and routing for a given flow layer. Extensions of that work have led to developments in a microfluidic description language similar to hardware description languages used in electronics (McDaniel et al., 2013), algorithms for laying out the flow layer based on a high level description of chip function (Minhass et al., 2012), and better algorithms for valve placement and control routing (Minhass et al., 2013; Tseng et al., 2013).

2.3 Using microfluidics to solve challenges in synthetic biology

Introducing complex engineered systems into cells presents numerous challenges at different levels of the synthetic biology workflow. I present in Figure 2·3 a sampling of the current challenges in synthetic biology and the microfluidic technologies most applicable to solving those challenges. I begin with the problems facing accurate specification of the function of novel genetic devices as the specification is only as useful as the understanding of the underlying biological behavior. A major challenge when designing new biological devices is that synthetic biology still suffers from a lack of well-characterized parts that do not interfere with each other when used to-

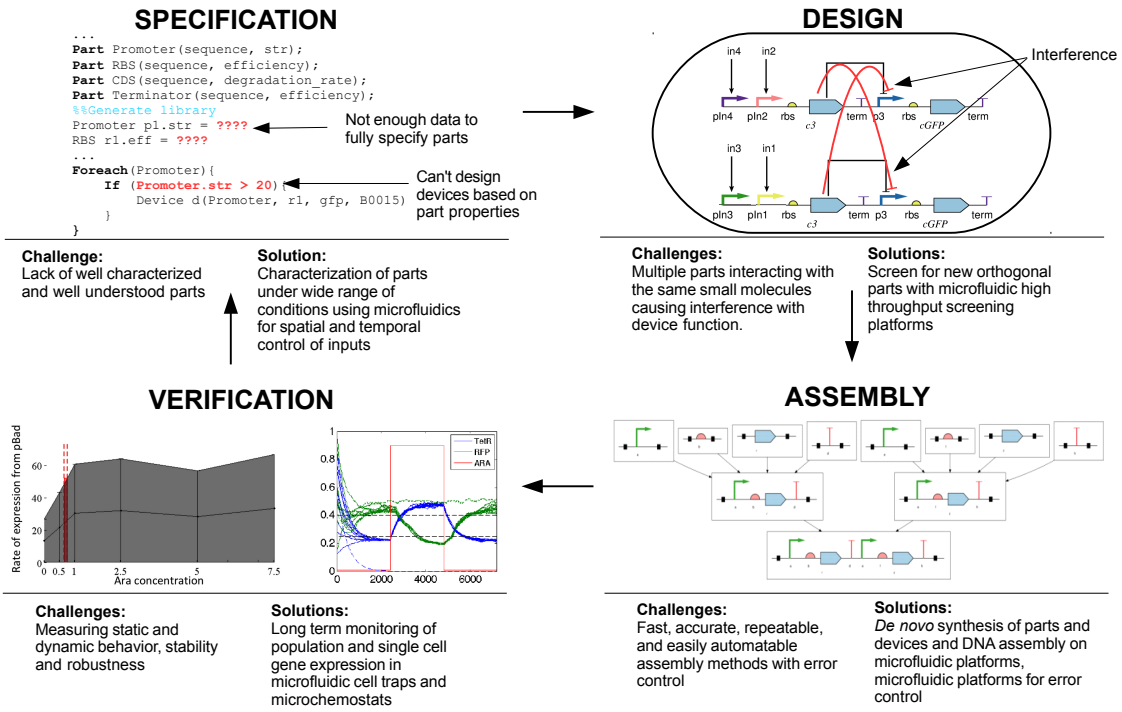


Figure 2·3: The synthetic biology workflow of specification-design-assembly-verification, and the challenges at each step in the workflow. The use of microfluidics in environmental control, high throughput screening, DNA synthesis, and cell culture may be used to augment current work in synthetic biology and address these challenges.

gether to construct larger systems. Important factors in the assembly of a device include potential unintended changes in function introduced when joining two segments of DNA and the time and cost efficiency of the currently available assembly methods. Verification of device function and stability requires accurate monitoring and measurement of protein expression at both the population and single cell level for extended periods of time. Microfluidics technologies in environmental control, high throughput assays, DNA synthesis, and cell culture can be used to augment and supplement existing work in synthetic biology to address these challenges. I present in Table 2.1 a summary of the current challenges facing synthetic biology as it matures

Workflow Domain	**Challenge**	**Potential Solution**	**Applicable Microfluidic Technology**	**References**
Specification	Accurate and standardized models of parts needed for composition of devices and prediction of device behavior in simulations	Characterization of parts and devices over wide range of environments and operating conditions	Large scale spatial and temporal fine-grained control over chemical inputs in microfluidic devices	(Dertinger et al., 2001; Lin et al., 2004; Bennett et al., 2008; Wang et al., 2012; Cooksey et al., 2009)
Design	Small number of available orthogonal parts result in unwanted molecular interactions in large devices	Engineering large libraries of new parts or reuse of existing parts	Microfluidic large scale integration for high throughput screening assays for new parts and controlling intercellular signaling for distributed biological computing	(Thorsen et al., 2002; Taylor et al., 2009; Dénervaud et al., 2013; Gómez-Sjöberg et al., 2007; Cheong et al., 2009b; Liu et al., 2010; Fidalgo and Maerkl, 2011)
Assembly	Fast, accurate, repeatable construction of large genetic devices with minimal interference of part/device function	*De novo* synthesis of genetic parts and devices	Microfluidic DNA synthesis and assembly	(Carr and Church, 2009; Kong et al., 2007; Lee et al., 2010; Huang et al., 2009; Kosuri et al., 2010; Kersaudy-Kerhoas et al., 2014)
Verification	Accurate measurements of gene expression at population and single cell level for multiple generations	Long term monitoring of single cells for part/device stability	Microchemostats and cell traps supporting long term cell growth and single cell monitoring	(Balagaddé et al., 2005; Ferry et al., 2011; Long et al., 2013; Danino et al., 2010; Locke and Elowitz, 2009)

Table 2.1: Key microfluidic technologies for investigation of challenges present in the synthetic biology workflow

as an engineering field and the most applicable microfluidic technologies that may be used to address those challenges.

2.3.1 Specification

Accurate specification of biological device function requires prediction of future behavior of combinations of biological parts and a standardized input/output model to share behavioral data across different designs. However, with the exception of some well studied systems such as the quorum sensing system (Collins et al., 2006), not all biological behavior in synthetic biological parts is well understood or predictable. In many cases, a device that performs well in one host system fails to perform when transplanted into a different host. In addition, different parts are characterized with different experimental methods such that there is no one standard of input/output measurement to specify interfaces between devices. As a result, many new devices fail to function without extensive trial-and-error, which is costly in both time and materials. To build more predictable systems, better characterization of part behavior over a wide range of environmental conditions is required to achieve the necessary understanding to build the correct models.

Challenges

In electrical engineering, simulation is used to predict device behavior and debug potential design flaws without spending time and resources in the lab. Whereas there is a solid understanding of semiconductor device physics and the ability to create accurate models for electronic parts for simulations, similar knowledge needed to create models of biological parts is still being developed. Chen et al. characterized terminator efficiency for 582 natural and synthetic transcriptional terminators and from that data generated a predictive model of terminator behavior given the sequence (Chen et al., 2013). Similar models have been generated for ribosome binding sites (Salis et al., 2009), but these sequence based models alone cannot predict the behavior of combinations of parts in a biological system. Likewise, new software tools developed

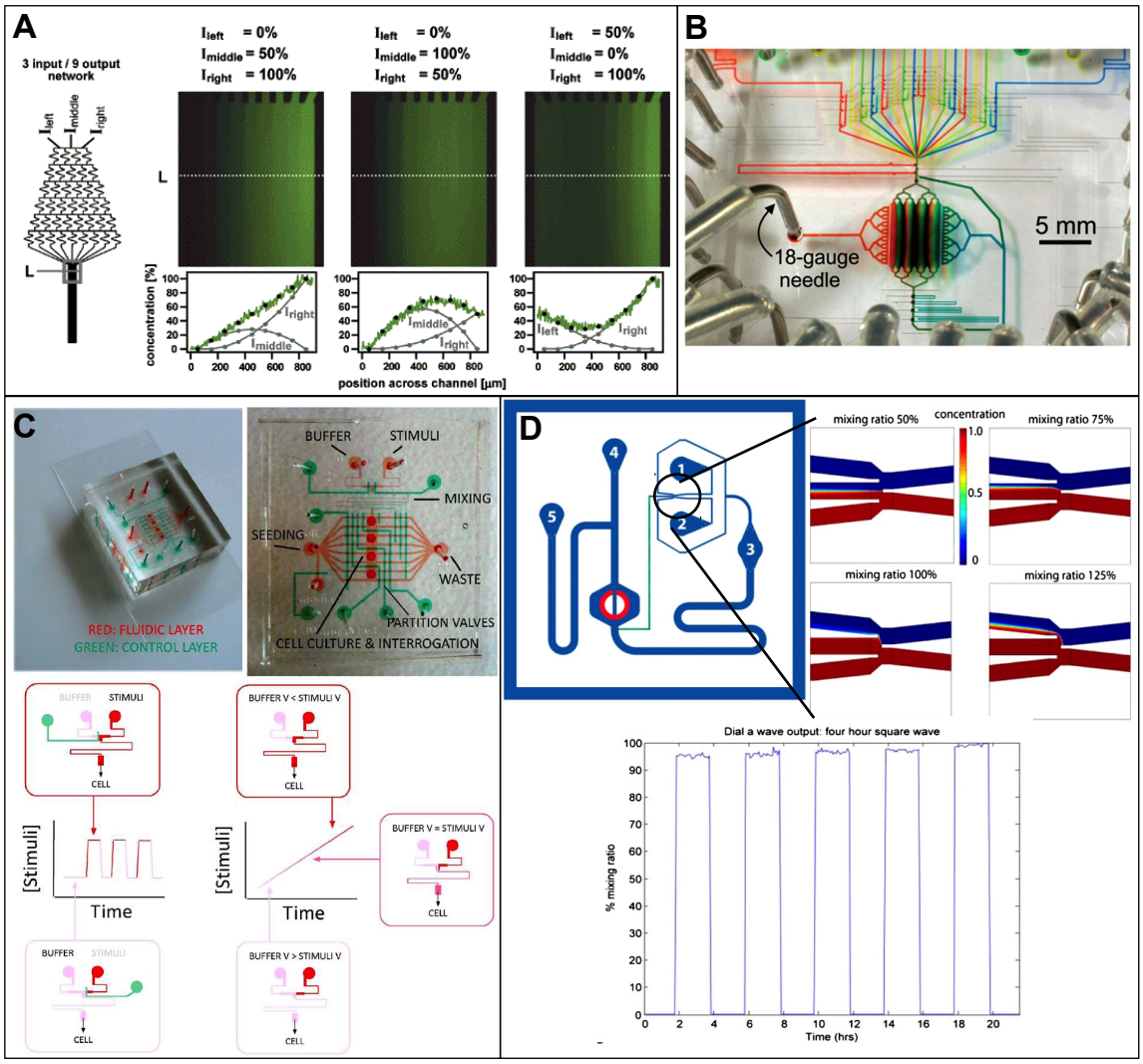


Figure 2·4: *(A)* Resistance network used to generate complex spatial gradients in fluid channels. (Reprinted with permission from S.K.Dertinger, D.T.Chiu, N.L.Jeon and G.M.Whitesides, Analytical Chemistry, 2001, 73, 1240–1246. Copyright (2001) American Chemical Society."). *(B)* Microfluidic chip for multipurpose testing using a combination of 16 inputs and outputs to generate spatial gradients in the central test chamber. (Reproduced from Ref. (Cooksey et al., 2009) with permission from The Royal Society of Chemistry.) *(C)* Dial-a-Wave device for generation of temporal stimuli via adjustment of flow interface for study of synthetic oscillators. (Used with permission from (Baumgartner et al., 2011). Copyright 2011 Buamgartner et al..) *(D)* Microfluidic function generator using valves to control flow through a T junction. (Used with permission from (Wang et al., 2012).)

for simulating synthetic biological systems (Jang et al., 2012; Madsen et al., 2012) are based on models of the biological processes of cell growth, diffusion, and protein interactions and degradation but may not take into account the details of part function based on DNA sequence.

Datasheets for electronic parts contain the information needed to create accurate behavioral models of those parts for use in simulations, including the valid input and output ranges, the switching characteristics, and frequency response for noise analysis. Canton et al. (Canton et al., 2008) postulated the creation of similar datasheets for biological parts and produced a datasheet for BioBrick BBa_F2620, a device that produces the transcription factor LuxR and is controlled by a regulated operator. Datasheets for biological parts must include different information than their electrical engineering counterparts due to issues such as host context and degradation rate of inputs and outputs that have no electronic parallels. Biological parts need to be characterized for orthogonality and multi-component behavior as well as the more usual single-component behavior. The long-term behavior of a part depends on such factors as the strain and growth stage of the host cell, the mutation rate, and environmental conditions such as temperature, pH, and culture media. For example, degradation rate of acylhomoserine lactone (AHL), the key signaling molecule in the popular and commonly used bacterial quorum sensing system, varies with both pH and temperature, making devices using this system sensitive to environmental changes (Kittleson et al., 2012). The addition of biological parts and devices into the host cell introduces large amounts of foreign DNA that may impact host cell metabolism (Klumpp et al., 2009). To produce useful datasheets and models for simulation would require massive amounts of characterization data. The spatial and temporal environmental control that microfluidics provides, combined with high-throughput cell culture assays (described in Section 2.3.2 could be used to develop characterization and test platforms

for synthetic biology devices.

Current solutions

Static analysis of biological devices involves determining the valid input and output ranges and generating the input-output curve that shows the range of outputs for any given input. Inputs to a biological device are transcription factors at various concentrations, and outputs are often fluorescent reporter proteins (Canton et al., 2008). A wide range of inputs is applied to the device under test to generate the full range of outputs. Additional characterization includes testing for interactions with other similar transcription factors and testing the effects of simultaneously applied multiple inputs.

Microfluidic resistance networks for generating complex spatial and temporal gradients (Figure 2·4A) (Dertinger et al., 2001; Lin et al., 2004) are well suited to generating the input ranges needed to characterize basic device behavior. Combining these networks with the ability to select multiple inputs through multiplexing allows for testing interoperatiblity and orthogonality (Lu, 2010). This technique has been used for combinatorial drug screening (Kim et al., 2012). The microfluidics chip by Cooksey et al. (Figure 2·4B) provides the ability to generate complex spatial gradients in a central chamber from combinations of up to 16 unique inputs by using segments of high fluid resistance and outlets to control flow (Cooksey et al., 2009).

Dynamic characterization requires the ability to trigger inputs on and off at a given frequency and measure the delay between the change in input and change in output as both valid and invalid inputs require time to propagate through the system. Faults may occur in a system when inputs are changed before the output is stable, when unstable outputs are used in a downstream function, or when the outputs of unstable inputs are used. Determining these timing characteristics of a biological part requires precise temporal control over the inputs to that system.

Wang et al. used valves to control inputs through a T junction to create square waves and ramp functions (Figure 2·4C) to characterize the dynamic signaling behavior of the social amoeba *Dictyostelium discoideum* as it transitions from a single-celled to a multicellular form during its life cycle (Wang et al., 2012). A different method of function generation uses laminar interface guidance to direct the laminar interface between input flows by adjusting the ratio of input flows (Bennett et al., 2008). Further refinement of this method led to the development of the 'Dial-a-Wave' device (Figure 2·4D) used to study the dynamics of environmental effects on the galactose metabolism network in yeast (Baumgartner et al., 2011). These microfluidic function generators can be also used for frequency domain analysis, which allows for applications of control theory techniques (Lu, 2010) such as block modeling and provides additional insights on noise and stability in biological devices (Cox et al., 2006; Simpson et al., 2003). Advances in microfluidic fabrication means that on-chip microfluidic oscillators (Mosadegh et al., 2010; Duncan et al., 2013) could also be used to generate complex input stimuli waveforms for device characterization while relying less on external hardware.

Future work

Currently, part characterization experiments for static and dynamic behavior are carried out individually, and as such, are costly in both time and reagents. The expense of these experiments no doubt contributes to the lack of available characterization data. As a next step in the progression of designs useful for part characterization, I suggest a microfluidics platform for multi-dimensional characterization of biological parts. Such a device would allow for simultaneous experiments and data collection of the input/output dose-response behavior, timing characteristics, and noise analysis through measuring single cell gene expression (Lu, 2010).

A starting point for such a platform could begin with a design similar to the

gradient generator by Cooksey et al. (shown in Figure 2·4B). A number of cell traps or microchemostats could be placed in the central chamber for monitoring cell growth and gene expression at the single cell level. A full discussion of single cell analysis in microfluidics is provided elsewhere (Yin and Marshall, 2012), while some of the key microfluidic devices in single cell trapping and cell culture are described in Section 2.3.4. The input to the central chamber could then be switched between a gradient generator and a waveform generator to allow for multiple types of experiments on the same device. Ideally, multiple experiments could be run on the same biological part simultaneously with the same microfluidics setup on this device.

For example, the biological part in this microfluidics device could be subjected to a gradient of inputs to measure the dose response. The inputs could be turned on and off at will to generate the temporal waveforms needed to measure the timing characteristics. Finally, as the cell traps could support single cell analysis, noise analysis could be performed on the part. Being able to perform many different experiments using one setup could allow for rapid characterization of new biological parts and devices.

2.3.2 Design

The nature of biology and evolution results in many homologous biological parts and pathways. Using homologous parts in a device leads to unwanted molecular interactions in the cell and interferes with the intended function of the device. These unwanted interactions are referred to as biological crosstalk. The likelihood of crosstalk increases as biological devices grow more complex and involve more regulatory networks (Voigt, 2006). The use of orthogonal parts (parts that do not interfere with each other) reduces crosstalk, but there is a lack of these parts in the current repertoire of synthetic biology. To increase the scalability of biological devices, new orthogonal parts and regulatory systems (Stanton et al., 2014) must be found or the currently

available parts and systems (Tamsir et al., 2011) must be reused.

Challenges

New orthogonal parts can be discovered by surveying known genomes for novel regulatory networks. By mining the genomic database at the European Bioinformatics Institute, Stanton et al. curated a collection of 73 homologs to TetR (a commonly used gene comprising of a repressible promoter and the repressor protein), and from those homologs, screened and isolated 16 orthogonal promoter-repressor pairs for use in new genetic devices (Stanton et al., 2014). The limiting step in this process is the final screening of pairwise interactions as over 5000 individual experiments were required to screen the homolog library. Microfluidic high throughput screening platforms would allow for hundreds, if not thousands, of parallel experiments and reduce the time needed to discover novel orthogonal parts.

New parts may also be obtained through directed evolution, a method of applying selective pressure to a library of variants to engineer for specific functions without prior knowledge of the system (Cobb et al., 2012). Conventional methods for cycles of mutation, cell growth, and selection require frequent human intervention and several days per cycle, but new automation techniques such as Multiplex Automated Genome Engineering (MAGE) (Wang et al., 2009) reduce both time and human attention required to generate large libraries. Using MAGE to optimize the pathway in *E. coli* that produced isoprenoid lycopene required screening approximately 10^5 colonies after 5-35 evolution cycles. The scale of microfluidic devices is too small to screen for the level of diversity produced by MAGE, but perhaps may be used as a secondary screening platform on a subset of the optimized colonies.

One method of part reuse, as demonstrated by Tamsir et al., is to separate large circuits into smaller circuits, each in a different cell colony, which communicate with each other through intercellular signaling chemicals. Using this technique, they built

all possible two-input Boolean functions from biological NOR gates. The colonies are spatially separated on a plate by hand, with the intercellular signaling chemicals spreading through diffusion. However, the diffusion of these signals is not directed towards specific colonies and may reach unintended targets to cause crosstalk between circuits. Microfluidics could be used to physically isolate each colony and restrict intercellular signaling to specific colonies by controlling media flow to reduce crosstalk.

Current solutions

An early device by Gómez-Sjöberg et al. contained 96 cell culture chambers, each with the capability for unique culture conditions, for automated screening and assays (Figure 2·5A). This device was used to study the effects of transient stimulation on human stem cells (Gómez-Sjöberg et al., 2007). Increased miniaturization of components and a focus on single cell imaging have led to higher density screening platforms. A device by Cheong et al. (Figure 2·5C) allows high throughput analysis of single cell signaling dynamics in 32 cultures (Cheong et al., 2009b; Cheong et al., 2009a). Taylor et al. developed a device with 2048 cell culture chambers capable of conducting 256 simultaneous screening experiments (Figure 2·5B) for studying mating hormone responses in yeast (Taylor et al., 2009) while Denervaud et al. developed a parallel microchemostat array for growing and observing 1152 strains of yeast-GFP strains (Dénervaud et al., 2013). These devices could be adapted for the screening and observation of part libraries developed through genomic mining and directed evolution.

An example of using microfluidics to control intercellular signaling is described by Liu et al. (Liu et al., 2010) with a four chamber network connected by thin channels for communications which could be opened or closed with valves (Figure 2·5D) used to study the response of NIH 3T3 fibroblasts to soluble signals from

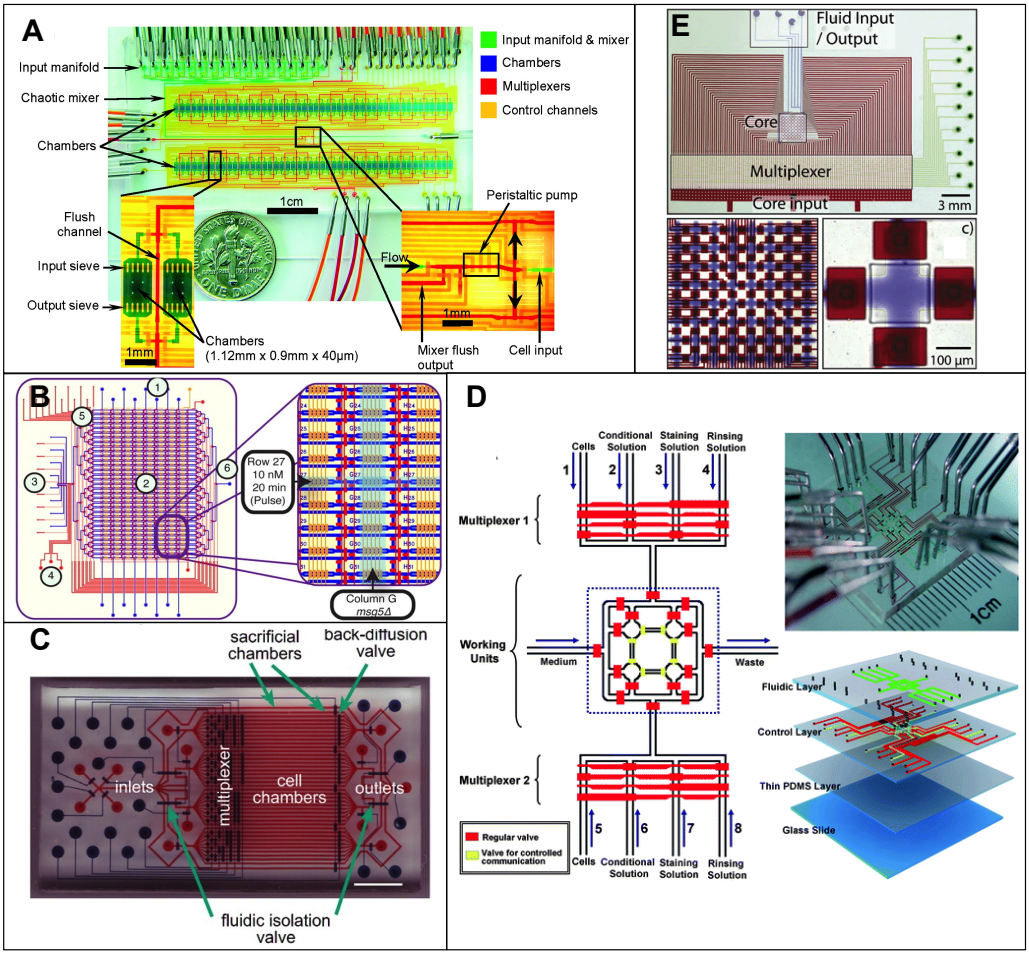


Figure 2·5: *(A)* 96 chamber cell culture device that allows for individual conditions in each chamber. ("Reprinted with permission from R. Gómez-Sjöberg, A. A. Leyrat, D. M. Pirone, C. S. Chen and S. R. Quake, Analytical chemistry, 2007, 79, 8557–8563. Copyright 2007 American Chemical Society.") *(B)* High throughput screening device capable of 256 simultaneous screening experiments. (Used with permission from (Taylor et al., 2009). Copyright 2009 Taylor et al.) *(C)* High content cell screening device used to study cell signaling. (Used with permission from (Cheong et al., 2009a). Copyright 2009 Cheong et al..) *(D)* Device for control and monitoring of intercellular communications. (Reproduced from Ref. (Liu et al., 2010) with permission from The Royal Society of Chemistry.) *(E)* Programmable general purpose microfluidic architecture with 64 nodes for reactions and liquid storage. Insets show the array of nodes and the valves surrounding each node. (Reproduced from Ref. (Fidalgo and Maerkl, 2011) with permission from The Royal Society of Chemistry.)

hepatocellular carcinoma cells. A network similar to this could be used to isolate cells and control signaling in biological distributed computing. A general purpose software-programmable architecture of an array of nodes surrounded by individually addressable valves similar to the one developed by Fidalgo et al. (Figure 2·5E) could be used to increase the size of circuits used in distributed biological computation. Incorporating basic Boolean logic directly into the chip through the use of pressure gain valves (Nguyen et al., 2012; Weaver et al., 2010; Devaraju and Unger, 2012) could allow further scaling of the microfluidic architecture by reducing the number of external control lines needed for larger experimental setups. Preliminary work has also been done on integrating microfluidic devices with liquid handling robots to increase automation and decrease the reliance on external control lines (Waldbaur et al., 2013).

Future work

The idea of distributed biological computing (Tamsir et al., 2011; Regot et al., 2011; Macía et al., 2012) can be expanded upon with high-throughput arrays and control of intercellular signaling through valves to produce a hypothetical platform for biological computing. This platform might leverage technologies already present in electrical engineering and digital design to further biological circuit design in synthetic biology. The architecture for this platform may contain both the microfluidic architecture and the software for integrating biological circuit design with microfluidic valve controls.

The microfluidics architecture for this hypothetical platform could contain banks of ports for inputs and outputs and an array of microchemostats to house the cells used for computing. Each port and microchemostat could be individually accessible through the use of multiplexing, and each microchemostat could house a cell colony containing a biological device for the basic unit of computation in the larger biological circuit. Chemical signals could be passed from one stage of computation to the next

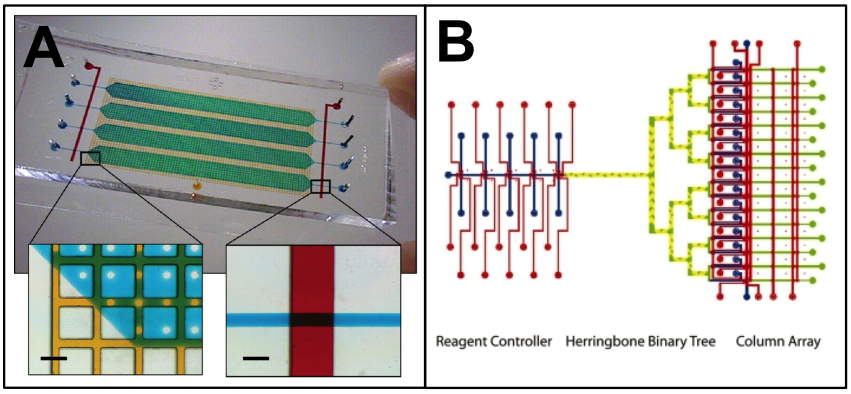


Figure 2·6: *(A)* Microfluidic chip capable of 4 parallel 500nL synthesis reactions, with the gene synthesis chamber shown on the left in yellow and green and the fluid channel and valve overlay on the right in blue and red. (Kong, David S, Carr, Peter A, Chen, Lu, Zhang, Shuguang and Jacobson, Joseph M, "Parallel gene synthesis in a microfluidic device", Nucleic acids research, 2007, 35, 8, e61, by permission of Oxford University Press) *(B)* 16 column microfluidic DNA synthesizer comprising of the reagent controller, a herringbone mixer, and a reaction column array. (Lee, Cheng-Chung, Snyder, Thomas M, and Quake, Stephen R, "A microfluidic oligonucleotide synthesizer", Nucleic acids research, 2010, 38, 8, 2514-21, by permission of Oxford University Press).

through the opening and closing of specific valves. An extra isolated microchemostat housing a biological oscillator circuit could be used to synchronize the other biological devices on the chip if the quorum sensing system is used for intercellular communications (Prindle et al., 2012). Synchronization could prevent logic faults caused by timing such as incorrect inputs being used in calculations or incorrect outputs being read.

2.3.3 Assembly

A vital part of synthetic biology is the technology used to assemble individual DNA parts into complex devices. The ideal DNA assembly method would allow for arrangement of parts in a specific sequence without scarring that would interfere with device function, easy generation of combinatorial libraries of constructs, and automa-

tion of the process (Cheng and Lu, 2012). However, modern assembly methods still fall short of the ideal.

Challenges

BioBrick Standard Assembly and its variants BglBricks and 2ab assembly (Leguia et al., 2013) depend on standardized flanking restriction enzyme cut sites for pairwise assembly of parts and create constructs with 8 base pair (bp) scars between parts. This class of methods is time-intensive for large constructs as each reaction requires several days, and automation has been limited by the sequential nature of the assembly process (Ellis et al., 2011). More promising are the one pot reactions such as GoldenGate (Engler et al., 2008), GoldenBraid (Engler et al., 2009), and MoClo (Weber et al., 2011), which allow for combination of multiple parts in a single reaction with smaller scar sites. However, both types of assembly methods use restriction enzymes to cut and join parts, so all parts used in these methods must first have all internal restriction enzyme sites removed through targeted mutations (Ma et al., 2012). This adds another step and another point of potential failure in the construction of large devices. *De novo* synthesis of complete devices could eliminate the problems with DNA assembly, but the price point of current DNA synthesis at \$0.40 to \$1.00 per base pair (Kosuri et al., 2010), with an increase in price to dollars per base pair for large constructs of thousands of base pairs (Goldberg, 2013), and inconsistent turn-around time (Shetty, 2013) makes the technology unsuitable for large devices. In addition, the price of DNA synthesis has plateaued at approximately \$0.40 per base pair, and is unlikely to decrease further without a a drastic improvement in technology (Goldberg, 2013).

Current solutions

As most of the price of conventional DNA synthesis comes from on reagent use and sample handling (Kong et al., 2007), the reduction in scale provided by microfluidic chips could lower the price of DNA synthesis to be viable for *de novo* synthesis of devices. An early example by Kong et al. (Figure 2·6A) consists of four 500nL parallel reactors, each capable of synthesizing genes of up to 1 kilobase (kb) in length from starting concentrations two orders of magnitude lower than conventional reactions. A similar device by Lee et al. (Lee et al., 2010) (Figure 2·6B) provides up to a 100 fold reduction in reagent use while producing 16 oligonucleotides in parallel at concentrations that did not require amplification before assembly. Combining this with the technology for the mass production of oligonucleotides in microarrays (Kosuri et al., 2010) and the development of microfluidic chips for two-step gene synthesis (Huang et al., 2009) could lead to an decrease in price of DNA synthesis to under $0.05 per base pair, and reach the point where synthesis of large constructs becomes viable. Microfluidic devices have also been developed for purifying synthesized oligonucleotides, which increases the fidelity of the final product (Kersaudy-Kerhoas et al., 2014).

Future work

While much work has been done in DNA sequencing and amplification (Hamon and Hong, 2013), little literature exists on importing existing DNA assembly techniques used in synthetic biology to microfluidic devices. Implementing processes such as BioBrick or MoClo assembly on microfluidic platforms would increase throughput and automation and decrease reagent use. Chang et al. (Chang et al., 2012) have proposed and patented a flexible microfluidic system for both pairwise and one-pot DNA assembly based on existing procedures for DNA ligation and amplification and cell transformation (Hong et al., 2006) in microfluidic systems. The company Genabler

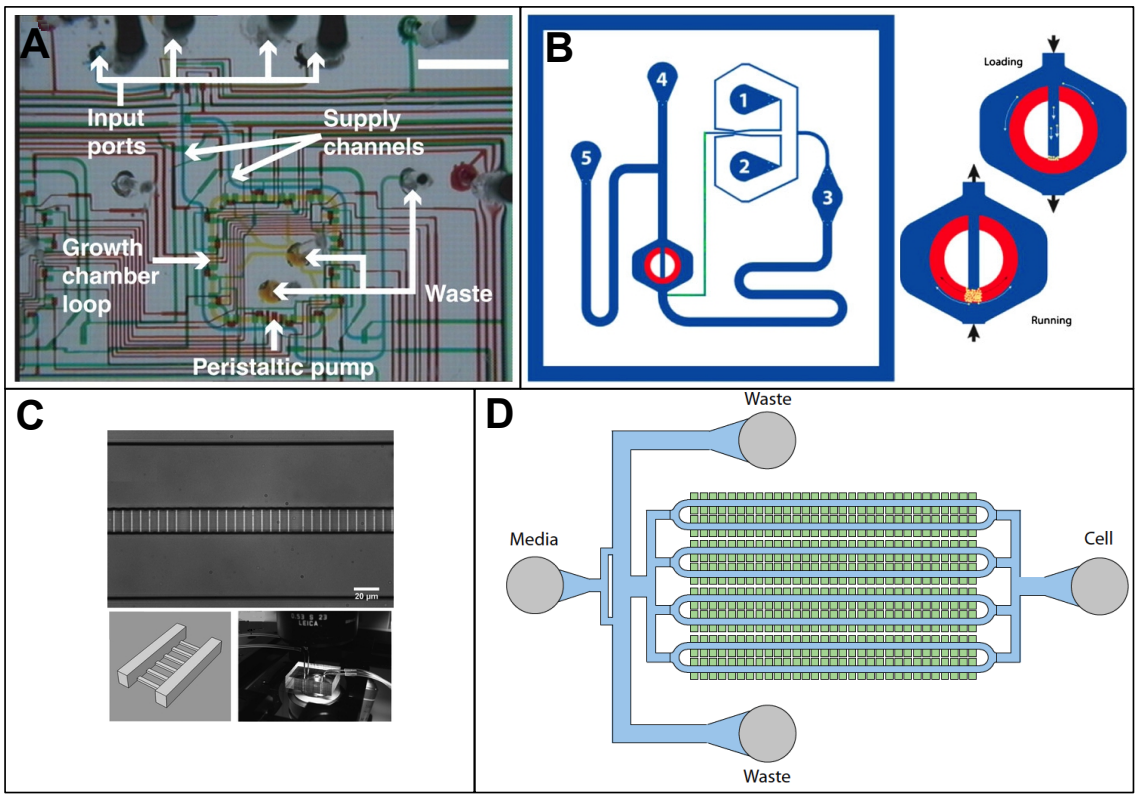


Figure 2·7: *(A)* Microchemostat for long-term study of *E. coli*. (Used with permission from (Balagaddé et al., 2005).) *(B)* Donut cell trap for study of yeast regulatory networks. (Used with permission from (Baumgartner et al., 2011). Copyright 2011 Baumgartner et al..) *(C)* Microchemostat that forces cell growth in lines for observation of single cell dynamics. (Reproduced from (Long et al., 2013) with permission from The Royal Society of Chemistry.) *(D)* Cell trap array for the study of synchronizing synthetic oscillators. (Reprinted by permission from Macmillan Publishers Ltd: Nature. (Prindle et al., 2012), copyright 2013)

has also developed a proprietary one-pot assembly method for use with a modular microfluidic system (Schmidt et al., 2013). However, little to no literature exists on the efficiency of commonly used DNA assembly techniques when performed in microfluidic devices.

2.3.4 Verification

The goal of verification is to compare the performance of the assembled device to the previously written specification, and from that comparison, determine the modifications needed to refine the design rules and models used in the initial specification. This requires subjecting the new device to a set of testing conditions and monitoring the expression of proteins of interest under those conditions. Whereas the behavior of individuals parts is the focus in previous sections, here the focus is on the analysis of systemwide behavior. The microfluidic devices described in Section 2.3.1 can be easily used to test the dynamics of novel biological devices and systems in addition to testing biological parts.

Challenges

One factor affecting the long term performance and reliability of biological devices not present in electronic devices is the natural mutation rate present in the host cell (Kittleson et al., 2012). Long term monitoring of biological devices via microchemostats and other similar cell culture setups is essential for characterization of the effects of mutation rate on the stability and robustness of novel biological circuits. Cell traps in these devices must allow for easy loading of cells and media, distribution of nutrients under high cell density, growing of cells in defined patterns to assist with tracking for single cell observations, and removal of cells without clogging the device (Ferry et al., 2011). Data at both the single cell level and the population level is required as important variations in single cells may be masked by the population data (Lecault et al., 2012). Monitoring device behavior at the single cell level also provides data on noise in the biological devices caused by random timing of biochemical reactions and discrete molecules, which is important for accurate modeling of circuit dynamics (Bennett and Hasty, 2009; Yin and Marshall, 2012).

Microfluidic platforms are well suited for and have been used in a variety of single cell studies (Zare and Kim, 2010), including investigating the mystery of antibiotic persistence in bacteria (Balaban et al., 2004). Behavior of the cells in these devices are monitored through time-lapse fluorescent microscopy (TLMF), and images of the cells are processed with segmentation algorithms to track cell lineage and gene expression at the single cell level(Locke and Elowitz, 2009).

Current solutions

Hasty et al. pioneered the use of microfluidics in the study of the dynamics of synthetic biological circuits and the synchronization and entrainment of genetic oscillators in *E. coli* (Danino et al., 2010; Mondragón-Palomino et al., 2011; Prindle et al., 2012). The main device used in these studies (Figure 2·7D) consists of a main feeding channel flanked by rectangular trapping chambers that are sized for the ideal distribution of cell density, nutrients, and signaling enzymes for intercellular oscillators (Danino et al., 2010) and allows for exponential growth of a colony for up to 4 days. Growing cells are pushed outside the chambers and swept away to the waste ports. For yeast studies, the Hasty lab has developed a microchemostat with a doughnut trap to control cell growth and the Dial-a-Wave junction (as described in Section 2.3.1) to control input flow (Figure 2·7B) that allowed cell growth for up to several days under various input conditions (Baumgartner et al., 2011). The construction of both this device and the version designed to optimize resource use with 8 parallel microchemostats capable of running individual experiments are described in detail as a case study of using microfluidics to study synthetic systems (Ferry et al., 2011).

The microfluidic bioreactor chip (Figure 2·7A) developed by Balagadde et al. was designed to inhibit biofilm formation and allow for long-term continuous cell growth in six 16nL microchemostats. This device was used to study the long term growth of *E. coli* programmed with a genetic kill switch regulated by cell density through quorum

sensing and the dynamics of a synthetic *E. coli* predator-prey system (Balagaddé et al., 2005). A microchemostat for constraining cell growth in lines for easy tracking (Figure 2·7C) developed by Long et al. consists of 600 sub-micron growth chambers connected to two feeding channels that trap *E. coli* and was used to observe growth rate and GFP expression at the single cell level (Long et al., 2013). These microfluidic chips, combined with complex input generation schemes and multiplexing, can be used to develop sophisticated setups for characterization of long-term static and dynamic behavior of biological parts and devices

Future work

One unexplored avenue in using microfluidics to study dynamics of gene expression is using feedback to influence and control the device of interest. Feedback may be used to stabilize device behavior, remove non-linearities, or account for system fluctuations, all of which would enhance the understanding of novel biological systems. Feedback systems already exist in microfluidic devices in the form of on-chip oscillators, and *in silico* feedback combined with optogenetics (Toettcher et al., 2011b) have been used to implement feedback systems regulating intracellular signaling in fibroblasts (Toettcher et al., 2011a) and gene expression in yeast (Milias-Argeitis et al., 2011). A new system combining *in silico* feedback with microfluidic control of gene expression could be useful in characterizing and studying system-level behavior of biological devices.

2.4 Hastening adoption of microfluidics

Despite these many applications for microfluidics in augmenting the synthetic biology workflow, adoption of microfluidics as a potential experimental and test platform has been slow in the synthetic biology community at large. Araci and Brisk. describe in a recent review on mLSI microfluidics the current shortcomings of the mLSI design

process that contribute to a lack of adoption of the technology (Araci and Brisk, 2014) . Chief among these are a lack of flexibility in the programming of devices and a lack of automated design tools. The majority of synthetic biologists lack both the expertise in fluid dynamics and microfluidic design and the expensive capital equipment for microfluidic fabrication. Layout of microfluidic designs by hand is both time consuming and error prone. In addition, experiments involving mLSI require a complex control platform including both software and hardware for valve control and fluid manipulation. Such setups are frequently customized for the application at hand and may be difficult to claim for future reuse. Improvements in automation for microfluidic device design and readily available open source software and hardware for control platforms could increase the speed of adoption of microfluidic technologies by synthetic biologists. This work seeks to remove the some of the barriers associated with microfluidics by improving automation of device design.

2.4.1 Design automation in microfluidics

Previously, CAD and automation have been primarily focused on droplet based digital microfluidics (Chakrabarty, 2010) rather than microfluidic large scale integration (mLSI). A new subset of tools are now being developed for layout and optimization of mLSI devices. The workflow for CAD for mLSI can be broken down as follows:

- A top level description language for both device design and function
- Placement of flow components
- Placement of control components
- Routing of flow and control components
- Optimization of control components, including minimization of the number of control pins, valve placement, and generation of control sequences

- Design rule checking and verification
- Output to a format usable for fabrication, i.e. vector graphics for photomask printing or 3D modeling file for 3D printing or machining

Earlier tools for mLSI include Biostream, a software tool for designing GUIs and control valves for multilayer devices (Thies et al., 2008; Urbanski et al., 2006) and Micado (Amin et al., 2009), which automated control valve placement and routing for a given flow layer. Recent developments have included a description language for microfluidic devices (MHDL) (McDaniel et al., 2013) which provides a framework for describing both the components of a device and rough guidelines for spacing.

For the placement of the flow layer, the simulated annealing algorithm (first used for VLSI placement) has been modified by several groups (Minhass et al., 2012; McDaniel et al., 2014) for use with microfluidic designs. Minhass et al. (Minhass et al., 2012) focused on schematic design and component allocation given a library of components and a functional description of the device while McDaniel et al. (McDaniel et al., 2014) focused on refinement of the simulated annealing algorithm for microfluidics. In particular, they described the effects of fixed placement of IO components and altering minimum spacing allotted around each component on both device design and algorithmic performance.

Similar to the placement algorithms, channel routing algorithms have also been derived from previous work on grid-based maze routers found in VLSI. Amin et al. (Amin et al., 2009) used an advanced min-cut max-flow algorithm for control channel routing while Minhass et al. (Minhass et al., 2012) and Hu et al. (Hu et al., 2014) used simpler maze routers based on Hadlock's algorithm and Lee's algorithm respectively.

The problem of control optimization was first discussed in Amin et al. (Amin et al., 2009), and expanded upon by both Minhass et al. (Minhass et al., 2013) and Hu et al. (Hu et al., 2014). Amin et al. inferred the placement of the valves from

Feature	**Micado**	**MHDL**	**Minhass et al. 2012**	**McDaniel et al. 2014**	**Hu et al.** 2014	**Fluigi**
Microfluidic Netlist Specification	no	yes	no	no	no	yes
Flow layer placement and routing	no	no	yes	yes	no	yes
Control layer placement and routing	yes	no	yes	yes	yes	yes
Automated control generation	yes	no	yes	yes	yes	yes
Design run checking	no	no	no	no	no	yes
End to end workflow	yes	no	no	no	no	yes
Open source	yes	no	no	no	no	yes
Compatibility with novel manufacturing methods	no	no	no	no	no	yes

Table 2.2: Comparison of current CAD tools for microfluidics

a description of the fluid flow in the device and routed valves to an optimized set of control ports, but the control ports had to be placed by hand. Minhass et al. further this by using a more advanced algorithm for control port minimization and automated placement of control ports. Hu et al. advance this further by optimizing control routing based on pressure propagation times and using a novel method of inferring valve placement.

Thus far, no automated tools for microfluidic design have touched on design rule checking and layout verification. It is assumed that the design parameters introduced during the placement and routing phases will prevent the layouts from violating established design rules.

2.4.2 An end-to-end automated workflow for microfluidic design

While there has been work on each step of the workflow for a comprehensive tool for microfluidic design, there has yet to be a program that produces a physical device

given an initial design specification. I present now a software tool, Fluigi, for the first end-to-end implementation of microfluidic design automation, from an inital design in the form of a microfluidic netlist to a fabricated microfludic device. Fluigi is an open source tool that provides automated placement and routing of both the flow layer and control layer given an initial list of components and their connections. For certain subsets of biological experiments focusing on using genetic devices to implement Boolean logic functions, Fluigi provides both automated device generation and automated control generation for the experiment. Finally, output designs from Fluigi may be used for both traditional photolithography and novel manufacturing methods such as 3D printing. Table 2.2 summarizes the capabilities of Fluigi as compared to currently available CAD tools for microfluidic design. Of note in particular is the fact that Fluigi is the only tool that provides design rule checking and interfaces to next-generation manufacturing methods.

Chapter 3

First forays into microfluidic CAD

I give in this chapter an overview of my earlier work on implementing a microfluidic CAD workflow. My earlier work focused primarily on using microfluidics to solve the problem of the lack of orthogonal parts in synthetic biology (Lu, 2010) rather than as a general purpose tool for designing microfluidic devices. The biggest changes between the two versions is the change in device architecture, which is further refined and formalized in the final version. As the initial concept was meant to be a dedicated platform for conducting biological computation in a microfluidic devices, the test cases focused on implementing biological Boolean functions of 2 and 3 inputs. Specifically, I used the 3-input XOR, 4-input AND, 8-3 encoder, and half adder as case studies. I also explored the possibility of designing microfluidic devices to implement all possible 2 and 3-input biological Boolean circuits with my workflow.

3.1 Motivation

Since logical frameworks such as repressible systems exhibiting the behavior of Boolean inverters already exist in gene expression systems, these frameworks may be repurposed to detect specific combinations of chemical or environmental triggers for targeted pharmaceutical and biotechnology applications (Tamsir et al., 2011). A summary of the various classes of genetic circuits for implementing Boolean logic functions in biology and their respective advantages and challenges is found elsewhere (Brophy and Voigt, 2014). I use the digital abstraction not to replicate silicon based com-

puting, but as an alternative method for designing robust biological circuits that are insensitive to noise and can be tuned to specific input conditions. The digital abstraction is well understood, and numerous techniques have been developed for its description (Micheli, 1994), synthesis (Weste and Harris, 2011), and verification (Hassoun and Sasao, 2002).

One application for biological logic circuits is in the field of cancer research, where the use of digital logic provides the necessary specificity for targeting strains of cancer cells while leaving other cells unharmed (Ruder et al., 2011; Shankar and Pillai, 2011). I show in Figure 3·1 a genetic `AND` gate in the cancer detection system described by Nissim and Bar-Ziv (Nissim and Bar-Ziv, 2010). This system uses two transcriptional start sites that are highly active only in cancer cells to detect the differences between tumor cells and premalignant cells. These two sites were chosen from the following: the chromatin structural protein histone-H2A1 promoter, the synovial sarcoma X-breakpoint protein-1 SSX1 promoter, and the inflammatory chemokine CXCL1 promoter. A targeting cell containing a logic circuit would be able to determine malignancy and deliver a payload of drugs or otherwise induce cell death only in tumor cells.

While devices implementing two-input Boolean logic functions are useful in synthetic biology, more complex computations involving more inputs would allow for applications such as the differentiation of molecular species, the identification of specific genetic markers, and environmentally tailored drug dosage responses (Purnick and Weiss, 2009). More complex functions may be constructed by increasing the levels of logic in the genetic device. A four-input transcriptional AND gate (Moon et al., 2012) with 11 orthogonal (non-interfering) regulatory proteins was constructed with this method from two levels of two-input AND gates. However, introducing complex systems such as this into a cell presents numerous challenges.

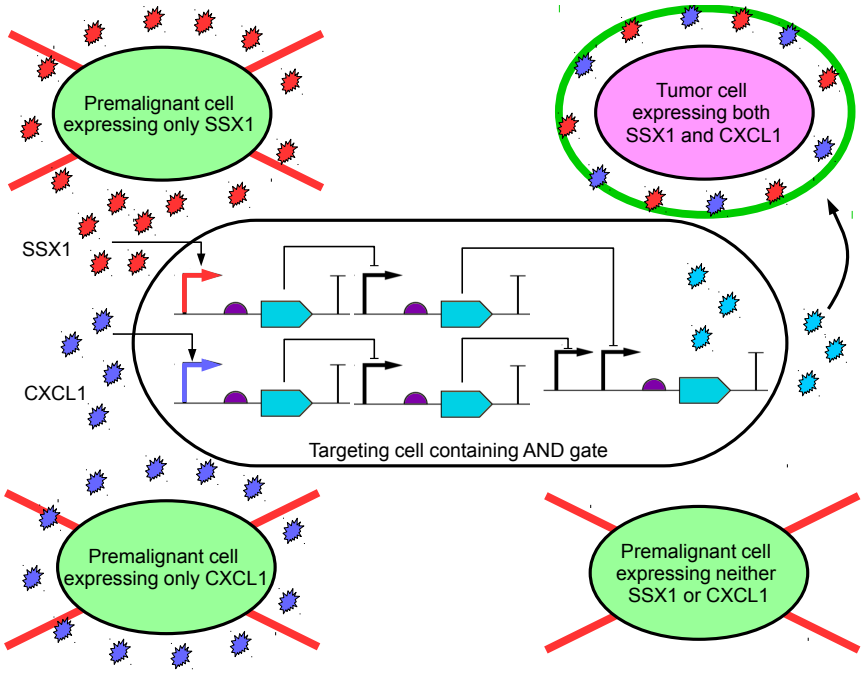


Figure 3·1: Two proteins highly expressed in tumor cells but not premalignant cells are the synovial sarcoma X-breakpoint protein-1 (SSX1) and the inflammatory chemokine (CXCL1) (Nissim and Bar-Ziv, 2010). A biological circuit implementing a Boolean `AND` gate was used to differentiate malignant and non-malignant cells based on their protein expression by only targeting cells that express both SSX1 and CXCL1.

A fundamental problem in building more complex biological systems lies in the lack of orthogonal parts and signaling molecules available (Moon et al., 2012) and the difficulty of engineering new orthogonal parts. Using non-orthogonal parts in the same system will result in unwanted molecular interactions in the cell which may interfere with the intended function of the system. These unwanted interactions are referred to as biological "crosstalk". As the number of unique regulatory proteins in the system increases, the number of factors to consider in the analysis and debugging of the system grows exponentially (Lu, 2010). In addition, the production of these regulatory proteins may also adversely affect cell metabolism (Klumpp et al., 2009).

These issues may be mitigated by distributing the computational load across many cells and separating large circuits into smaller circuits, each in a different cell colony,

that communicate with each other through intercellular signaling chemicals (Tamsir et al., 2011; Regot et al., 2011; Macía et al., 2012). Tamsir et al. built all possible two-input Boolean functions from biological `NOR` gates in this manner. The colonies are spatially separated on a plate by hand, with the intercellular signaling chemicals spreading through diffusion. However, the diffusion of these signals is not directed towards specific colonies and may reach unintended targets and cause crosstalk between circuits. Increasing the number of intercelluar signals to use in these systems does not fully address the problem of scalability as there is a lack of orthogonal intercellular signaling systems. I expand on the idea of distributed biological computing by using microfluidics to physically isolate each colony and restrict intercellular signaling to specific colonies via controlling the media flow to reduce this crosstalk.

The first step in creating a microfluidic platform for distributed biological computation is to build the necessary framework and tools to allow for easy iteration and refinement of my microfluidic design while taking into account the biological components. While rudimentary CAD tools for both microfluidic chips based on multilayer soft lithography (Amin et al., 2009; Minhass et al., 2012; Minhass et al., 2013) and synthetic biology circuit design (Xia et al., 2011; Cai et al., 2010; Chandran et al., 2009) exist, there are currently no CAD tools that combine, formalize, and optimize the two processes. This thesis presents a microfluidic design framework which allows for simple cellular synthetic biological systems to communicate in a controlled fashion while keeping the cell types physically separated. By abstracting the biological details ((Endy, 2005; Andrianantoandro et al., 2006)) I focus primarily on basic Boolean algebraic biological computing elements and an intercellular signaling fluid. Assuming that individual genetic circuits representing Boolean algebraic functions can be put into cellular systems (Moon et al., 2012; Tamsir et al., 2011; Daniel et al., 2013; Brophy and Voigt, 2014) and that intercellular communication mechanisms can be

combined with these systems (Bonnet et al., 2012; Basu et al., 2005) challenges still remain in using these systems on a large scale. A summary of some of the challenges and potential solutions are shown in Figure 3·2.

A strength of this approach is that I can use existing paradigms in the electronic design automation community and apply them to microfluidics. My results show that large sets of Boolean algebraic functionality can be realized with an automated software workflow using primitive genetic gates organized on a generic microfluidic valve-based fabric. These systems should prove effective at removing the described crosstalk issues, which will allow for more formally engineered systems. This will also allow me to leverage existing CAD infrastructure and tools. This paper focuses on describing how a design synthesis workflow, starting from a high level description of desired functionality, can be used to place the biological components in a microfluidic system, route the signaling fluid, control the valve network, and simulate the expected control system behavior.

3.2 First iteration

In my first attempt, I attempted to model a microfluidic device architecture using an infrastructure influenced heavily by FPGA design as a starting point. my basic architecture consisted of a grid of standardized 'tiles' that would house biological components and a border of input/output ports. Unfortunately, this proved to be a too simplistic and naive view of both biology and microfluidics as I failed to consider the problems of control layer routing and the upkeep of biological circuits.

3.2.1 Architecture

I begin by defining M to be a set of biological signaling molecules, C to be the set of available biological circuits, and G to be the set of all 2 input Boolean functions.

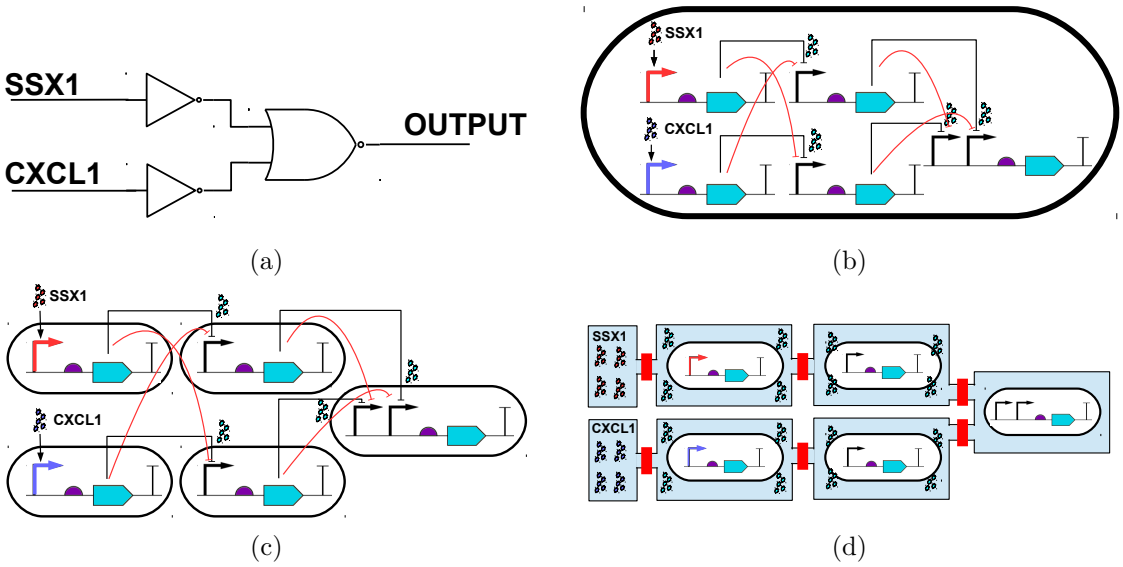


Figure 3·2: Figure 3·2a shows an `AND` circuit made of `NOT` and `NOR` gates and Figure 3·2b shows the same function implemented by genetic circuits in a single cell. Bent arrows represent transcriptional start sites, and blue arrowed boxes represent genes. Black arcs indicate repression of the transcriptional start sites by the proteins produced while red arcs indicate potential crosstalk sites between the proteins. Separating the circuits into multiple cells will lessen the metabolic impact of the circuit on the host cell, but only shifts the problems arising from crosstalk to the intercellular signals (3·2c). The use of microfluidic chambers and valves to physically separate the genetic circuits (3·2d) eliminates crosstalk and allows the precise application of external control molecules.

I define a `VALVE` as a double (M_s, s) where $M_s \subset M$ is the set of signals I expect to pass through this valve and state $s \in \{\texttt{OPEN}, \texttt{CLOSE}\}$ is the current state of the valve.

I define a `CHAMBER` as a quintuple $(C_0, M_i, M_o, V_i, V_o)$ where

- $C_o \subset C$ is the set of biological circuits present in the chamber
- $M_i \subset M$ is the set of input signaling molecules
- $M_o \subset M$ is the set of output signaling molecules
- V_i is the set of input valves to the chamber

- V_o is the set of output valves from the chamber

From those two definitions, I define a `NETWORK` as a non-empty set of valves V and a non-empty set of chambers H with the property such that each valve in the network must connect to some chamber in the network. I implement a network as a directed acyclical graph where the set of vertices is the set of valves V and the set of edges can be mapped to the set of chambers H where each chamber in H represents the set of edges between the valves in the set V_i and the valves in the set V_o.

I then define a `TILE` as a triple (n, g, c) where n is a `NETWORK`, gate $g \in G$ is the Boolean function that the tile implements, and c is a double (x, y) where x and y are non-negative integers that describe the location of the tile.

Finally, I define a `CHIP` as a directed acyclical graph where the set of vertices is a finite non-empty set of tiles T. I implement a chip as a grid of n x m tiles where each tile is referenced by its position in the chip by a double (x, y). Within a chip, I declare the set of input tiles T_i as the set of border tiles on the left, top, and bottom, and the set of output tiles T_o as the set of border tiles on the right. I assume all tiles in T and not in T_i have identical networks. An image of a hypothetical chip and its abstraction levels is show in Figure 3·3.

3.2.2 Workflow

The design flow begins with an input file, which is then converted to a Boolean algebraic canonical form such as a binary decision diagram or a truth table. The logic functions are minimized, and mapped to gates of the selected technology. Genetic circuits are selected to implement the functionality of the gates. The gates are mapped to the tiles of the chip, and genetic circuits are assigned to the chamber networks in the tiles. The chip is then routed, and the sequence of valve controls is generated from this layout. Fluigi also generates code to interface the valve controls to the existing

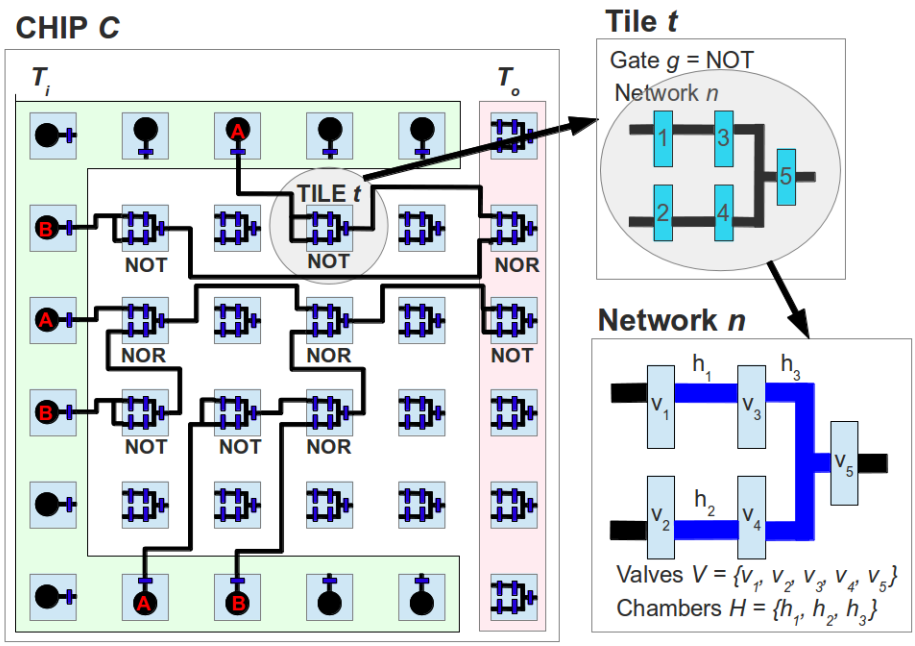


Figure 3·3: Hypothetical chip structure for "Fluigi". The chip C is composed of some set of tiles T. Input tiles T_i are highlighted in green, and output tiles T_o are highlighted in red. A tile $t \in T$ has a network n of a set of valves V and a set of chambers H.

microfluidics control software. Finally Fluigi creates an image of the chip layout that may be used as a photomask for microfluidic chip fabrication.

Fluigi extracts the Boolean functions from an input file and converts that function to a canonical form. It then minimizes the logic functions if appropriate and maps the functions to a selected technology. This can be done with a program such as Espresso (Hayes, 1993) or the Cadence Encounter design software starting from a Verilog file.

At this stage, I select a subset of gates $G_m \in G$ and represent the input functions as functions composed entirely of gates g in G_m. I construct a graph of the input functions R with gates g as the vertices and the connections between the gates as edges. Fluigi supports technology mapping to three subsets of G:

- $G_{NOR} = \{\texttt{NOR}, \texttt{NOT}\}$
- $G_{NAND} = \{\texttt{NAND}, \texttt{NOT}\}$

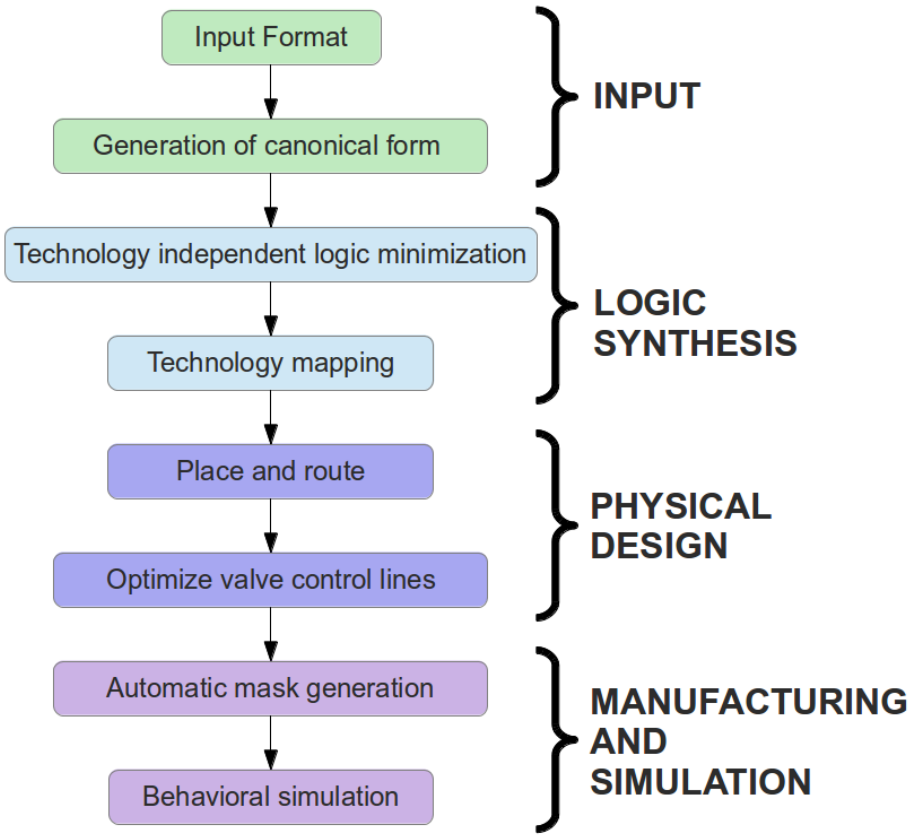


Figure 3·4: Proposed design flow for Fluigi starting with an input file and ending with an interface to existing microfluidic control software and chip simulation.

- $G_{MULTI} = \{$`NOR`, `NOT`, `NAND`, `AND`, `OR`$\}$

For each gate in G_m, I select a genetic circuit c in C that implements the functionality of that gate. Gates are duplicated to reduce fanout as molecular signals from biological components may not be concentrated enough to drive more than one input at a time.

Once I have a graph of gates R, I map the gates in R to tiles T in the chip C. When a gate g is mapped to a tile t, I place the genetic circuit selected for g in the chambers of the network of tile t. I first place the input and output gates in the appropriate tiles on the border of the chip and then randomly place the remaining gates in tiles in the center of the chip. I then create an edge between tiles in T for

each edge that exists in R. After the initial placement is complete, I calculate a score of the placement as the sum of the Manhattan distance between all connected gates. I minimize the score by pairwise swapping of gates in each region of the chip and then attempt to route the tiles in the chip using Lee's algorithm for maze routing. If the routing fails, I generate another possible placement and iterate the process until a viable routing is found.

From the routing, I generate code needed to control the chip with the existing microfluidics control software suite. As the existing control setup only has 48 available control lines for opening and closing valves, I first minimize the number of control lines my chip needs by pipelining both the tiles in the chip and the chamber network in the tiles to generate a mapping of valves to control lines. I then generate a set of four `Java` files required to interface to the existing microfluidics CAD platform BioStream, developed to design GUIs and control valves for multilayer devices (Amin et al., 2009; Thies et al., 2008; Urbanski et al., 2006) (freely available at (Thies et al., 2009)), and an image of the chip layout which may be used to generate a photomask for fabrication.

I separate the chip into pipeline stages of tiles by running breadth-first search starting at the output tiles and and saving the tiles at the same depth in a list such that I end with a list of lists of tiles. I next separate the network in a tile into pipeline stages in a similar manner by running breath-first search starting at the output valve. In this case, I also take into account the order that valves in the same stage must be opened as defined by the genetic circuit in the network of the tile. When I add new valves to a pipeline stage, I sort those valves in the partial order they must be opened. Valves with dependencies are shifted to other stages to preserve those dependencies. I create a valve pipeline for each different network I have in the system.

I generate the final mapping of valves to control lines by going through the pipeline

of tiles, and for each stage, getting the list of tiles in that stage. I then go through the pipeline of valves for those tiles and assign all valves in those tiles at the same stage of the valve pipeline to the same control line. I repeat this for all stages in the tile pipeline.

3.2.3 Results

I describe now two simple test tiles I used to demonstrate the features of Fluigi. The input tile consists of one valve and a reservoir of some input signal. Opening the valve allows the input signal to flow to the rest of the chip. The base tile, as shown earlier in Figure 3·3, consists of five valves and three chambers. At this time, I only consider control lines for the data path and not for flow of media or waste. I define that chambers h_1 with input valve v_1 and output valve v_3 and chamber h_2 with input valve v_2 and output valve v_4 contain buffer circuits used to amplify the incoming signal. Chamber h_3, with input valves v_3 and v_4 and output valve v_5, contains a synthetic biological circuit of a `NOR` gate such as the one described in (Tamsir et al., 2011). To prevent signal crosstalk, only one of v_3 or v_4 may be open at a time. My chip is then a grid composed of these two tiles. I assume a naive assignment of control lines to be one in which each valve is assigned its own control line such that n tiles a naive assignment have $5n$ control lines

I test the functionality of Fluigi with two sets of benchmarks. The first is all two-input Boolean logic functions, and the second is all three input Boolean logic functions. Two-input Boolean logic functions are a mainstay in synthetic biology as they provide the basic building blocks to more complex functions. Genetic circuits for these functions have been built and tested (Tamsir et al., 2011) and can be obtained. The second set of benchmarks demonstrates the generalized functionality of Fluigi on more complex functions. In addition, I test Fluigi on four specific example circuits: 1) the 4-input `AND` gate, the genetic circuit of which is described in (Moon et al.,

Table 3.1: Results of Fluigi for two-input benchmark circuits

Chip Size (tiles x tiles)	Number of Two Input Circuits	Average Tile Usage (%)	Average Unoptimized Control Lines	Average Optimized Lines	Average Line Reduction (%)
4x4	4	23.44	10.75	8	25.89
5x5	4	21.00	18.25	12	33.76
6x6	2	27.78	34	17	50.00

Table 3.2: Results of Fluigi for three-input benchmark circuits

Chip Size (tiles x tiles)	Number of Three Input Circuits	Average Tile Usage (%)	Average Unoptimized Control Lines	Average Optimized Lines	Average Line Reduction (%)
4x4	12	23.44	10.75	8.00	25.89
5x5	49	27.59	23.24	15.02	35.03
6x6	125	34.53	43.18	20.81	51.09
7x7	57	41.28	72.05	28.86	59.45
8x8	5	42.50	100.00	29.80	70.22

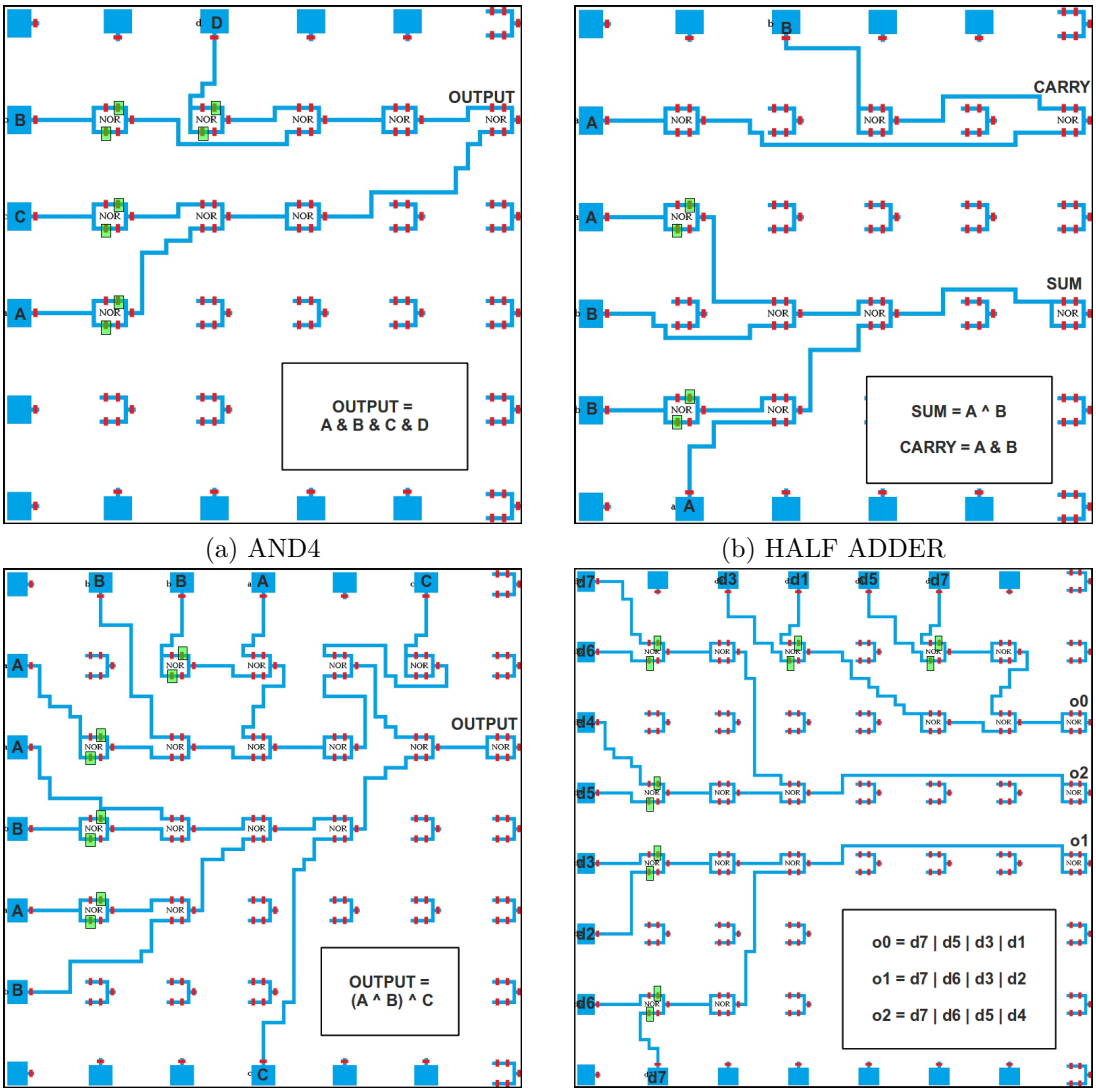
Table 3.3: Results of Fluigi for example circuits

Circuit	Chip Size (tiles x tiles)	Gates Used	% Usage	IO Used	% Usage	Control Lines	Optimized Control Lines	% Line Reduction	Average Channel Length (px)	Variation (px)
AND4	6x6	11	36.11	5	25.00	44	17	61.36	176	96
HALF ADDER	6x6	15	41.67	8	40.00	51	17	75	244	144
XOR3	7x7	26	53.06	11	45.83	90	29	67.78	219	120
8-3ENC	8x8	30	46.88	15	53.57	102	17	83.33	227	156

2012), 2) the half adder, a simpler version of the genetic circuit described in (Beal et al., 2012), 3) a 3-input `XOR` gate, as a representative of the three-input Boolean logic function benchmarks, and 4) an 8-to-3 encoder, the genetic circuit of which is being developed in the Densmore lab.

Benchmark circuits

The results of Fluigi on the benchmark circuits are shown in Tables 3.1 and 3.2, with results for two-input Boolean functions shown in Table 3.1, and the results for three-input Boolean functions shown in Table 3.2. When running the benchmarks,



(a) AND4 (b) HALF ADDER

(c) XOR3 (d) 8 to 3 Encoder

Figure 3·5: Routing of selected circuits as generated by Fluigi. These images serve as the GUI for simulation of high level chip behavior and may be used to generate the photomask for chip fabrication. Solid blue rectangles are input wells. Blue lines are channels and chambers, and red rectangles are valves. Valves highlighted in green in a chip are a set of valves that is activated by the same control line.

I ignored any obvious circuits with 0 or 1 inputs as those circuits are not useful for demonstrating the capabilities of Fluigi. I ran Fluigi on a total of 10 two-input Boolean functions and a total of 248 three-input Boolean functions.

All two-input Boolean functions could be placed on a chip of 6x6 tiles, and all

three-input Boolean functions could be placed on a chip of 8x8 tiles, 75% of which fit on a chip of 6x6 tiles. The average percent usage of the chip for all two-input Boolean functions was 23.33% while the average percent usage for all three-input Boolean functions was 34.33%. In both cases, the chip was underutilized due a combination of an unoptimized place-and-route algorithm and the restriction of input and output tiles to the border tiles. As all the benchmark functions only have one output, the majority of the output tiles were not used. Discounting the superfluous output tiles results in an average usage of 28.00% for two-input Boolean functions and 39.85% for three-input Boolean functions. Usage rates can be increased further through the use of a more sophisticated place and route algorithm, and underutilization can be fixed with function-driven masks.

Example circuits

I selected four circuits, the four-input `AND` gate, the three-input `XOR` gate, the half adder, and the 8 to 3 encoder, to describe and examine in further detail. For these four circuits, I compiled the code Fluigi generated and successfully ran chip-level behavior simulations of these circuits using existing microfluidics control software. The layouts for these four circuits as generated by Fluigi are shown in Figure 3·5, and the statistics for these are shown in Table 3.3.

In these layout diagrams, I draw the fluid chambers and channels in blue and the valves in red. Length of the channels is currently measured in pixels, which I can then convert to physical distance based on manufacturing technology for fabrication. I do not draw the control lines as my program does not yet take into account the routing of control lines. The valves highlighted in green in each chip are a set of valves tied to the same control line. In all four chips, the total number of optimized control lines needed for each chip is less than the maximum capacity of the control hardware.

Figure 3·5a shows the microfluidic version of the `AND4`, which uses 11 tiles, with

four input tiles and one output tile, and requires 17 control lines. Of the four example circuits, it has the shortest average channel length (176px) and lowest channel length variation (96px). Figure 3·5c is shown here as an example of a three-input Boolean function used previously as a benchmark circuit. This circuit required 26 tiles, with 10 input tiles and one output tile. Like the `AND4`, it is a well balanced circuit with comparatively low variation in channel length.

Figures 3·5b and 3·5d show Fluigi's ability to place and route logic functions with more than one output. Both circuits were successfully routed, the half adder on a chip of size 6x6 and the encoder on a chip of size 8x8. The half adder is a simpler version of the genetic arithmetic circuits first described in (Beal et al., 2012). It is composed of two smaller circuits, an `AND2` gate and an `XOR2` gate. In my implementation it requires 15 tiles, of which six are input tiles and two are output tiles. The number of optimized control lines for the circuit is 17, a 75% reduction for the original 51. The half adder has the highest average channel length due to the suboptimal placement of the tiles in the carry subcircuit.

The encoder consists of three identical four-input `OR` gates, requiring 30 tiles with 12 input tiles and 3 output tiles. Since the three subcircuits have the same structure, control line optimization is highly effective here, resulting in an 83.33% reduction in required control lines from 102 to 17. However, suboptimal placement of gates in tiles has led to this circuit having the highest variation of channel length. This can be fixed by altering the placement algorithm to minimize channel length variation as an additional constraint.

These layouts represent a good first approximation for photomasks for microfluidic chip fabrication. However, two key optimizations will increase the likelihood of creating viable photomasks. In this version, my place-and-route algorithm does not take into account the number of bends in each channel or the variation in channel

length. The algorithm will be improved by adding a bend penalty during routing and by altering the scoring function during placement to take into account average channel length.

3.2.4 Flaws

The biggest flaw with this architecture was the naive view of both microfluidics and biology it incorporated. The tile design did not leave room for introduction of media or the removal of waste to ensure growth of the cells containing biological circuits. In addition, I did not take into account the physical structures needed to control all the valves in a tile, and the architecture lacked both space for routing control channels and placement of control ports. While this demonstrated a first pass at a proof-of-concept workflow, more work remained to define an architecture that would result in a viable microfluidic device.

3.3 Second iteration

(This section originally published as (Huang and Densmore, 2014a).)

After some thought, I revised the architecture to represent a microfluidic device as a set of graphs, where each graph in the set represented a different layer in the device. This allowed me to automate the generation of photomasks for a set of devices for biological computation. However, the exact design was generated heuristically, and the design process was not easily extensible to a wider range of microfluidic structures as I limited the set of microfluidic features to ports, valves, channels, and cell traps.

3.3.1 Architecture

I define a `GATE` to be a group of cells that perform a specific function and have some number n of inputs and one output. For the examples shown in this paper, $n = 1$ or

$n = 2$. The top level function I am trying to implement can then be defined as GATEs connected in a directed graph.

I define a CHIP as having two elements, a flow layer and a control layer. The flow layer represents the paths fluids can travel on the chip, and the control layer represents the valves used to manipulate fluid flow and the connections between them. The flow layer can be represented as a directed graph and the control layer as an undirected graph. I assume both these graphs are sparsely connected. A hypothetical flow layer graph and control layer graph for a sample chip is shown in Figure 3·6a.

For the flow layer, I define a vertex in the graph to be a NODE and an edge in the graph to be a CHANNEL. A CHANNEL represents a segment along which air, fluid, or cells may travel. A subset of channels are CELL TRAPs, where GATEs may be placed in the chip. A NODE has one or more channels entering or exiting it, and must be connected to at least one channel. A subset of NODEs are FLOW PORTs, which represent locations where the chip interfaces with the outside world. Flow ports may be sources, which have no incoming channels or sinks, which have no outgoing channels. Fluids enter the chip through source ports and exit the chip through sink ports.

For the control layer, I define a vertex in the graph to be a VALVE and an edge in the graph to also be a CHANNEL. A VALVE is a location on the control layer that intersects a channel in the flow layer with the intention of manipulating flow in the channel. Each valve is associated with a channel in the flow layer. A subset of valves are the CONTROL PORTs, which are locations where air enters and exits the chip. Unlike flow ports, air may enter and exit the same control port. Each valve must be connected to a control port through a channel, and each control port must be connected to at least one valve. A summary of these definitions is found in Table 3.4.

Table 3.4: Definition of terms

Term	Symbol	Definition
CHIP	H	{FLOW LAYER, CONTROL LAYER}
FLOW LAYER	G_{FL}	directed graph $G_{FL} = (N, C)$, with N, a set of NODEs, as vertices and C, a set of CHANNELs, as edges
CONTROL LAYER	G_{CL}	undirected graph $G_{CL} = (V, C)$, with V, a set of VALVEs, as vertices and C, a set of CHANNELs, as edges
NODE	n	vertex in G_{FL}, physical location on the FLOW LAYER where one or more channels start or terminate
FLOW PORT	n_p	set of FLOW PORTs $N_p \subset N$, physical location where flow channels interface to locations off-chip
VALVE	v	vertex in G_{CL}, physical location where a control channel can be pressurized to create a seal over a flow channel
CONTROL PORT	v_p	set of CONTROL PORTs $V_p \subset V$, physical location where control channels interface to locations off-chip
CHANNEL	c	edge in G_{FL} and G_{CL}
CELL TRAP	c_t	set of CELL TRAPs $C_t \subset C$, physical location on a chip where cells may be placed
GATE	g	cells computing a function with $n > 0$ inputs and 1 output

3.3.2 Workflow

The basic unit of architecture is the flow stage, which has a single signal input node, one cell sample and one media input source port, one block of cell traps, a waste output sink port, and a single signal output node. A diagram of this is shown in Figure 3·6a, and the diagram for the control layer graph and flow layer graph for such a chip is shown in Figure 3·6a. The full physical design process from flow layer layout to control layer routing is shown from Figure 3·6b to Figure 3·6d. I use the cell trap design described in (Prindle et al., 2012) for my cell trap block. The flow stage unit may be expanded to contain more cell types by increasing the number of cell trap blocks and cell sample inputs. Flow stages may be linked together by connecting the output node of one stage to the input node of the next stage. This architecture gives

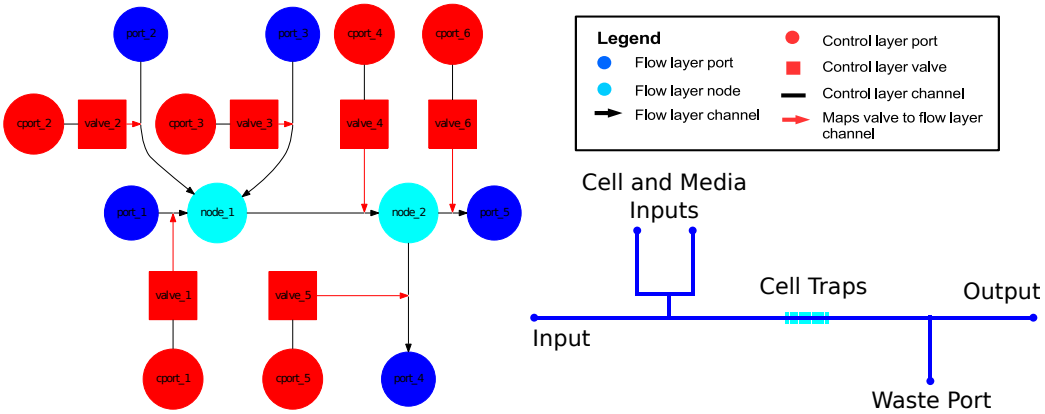
Table 3.5: Flow layer and control layer layout parameters

Parameter	Length (μm)
Flow channel width	100
Control channel width	50
Max. control channel width when crossing flow channel	20
Flow Port Radius	100
Control Port Radius	100
Valve width	2*Flow channel width
Valve length	Flow channel width
Min. distance between ports	1000
Min. distance between port and channel	400
Min. distance between port and valve	1000
Min. distance between channels	25
Min. distance between valve and channel	25

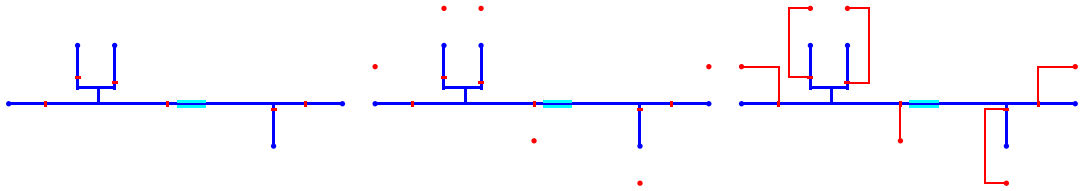
me the flexibility to make arbitrary-sized multi-stage circuits.

I assign valves such that each cell trap, source port, and sink port may be accessed individually and that each stage may operate independently. By connecting the cell traps and ports to be accessed in a binary tree (Thorsen et al., 2002), I can reduce the number of control lines needed to individually access those features. Using multiplexing, only $2\log_2 n$ control lines are needed to access n samples. I apply multiplexing when the number of samples to be accessed is greater than or equal to four. Otherwise, I assign control lines individually.

During layout, I use the parameters shown in Table 3.5 to determine physical layout constraints (Amin et al., 2009) with the most important constraints being the minimum distance required around ports and the minimum spacing around channels and valves. Control valves are sized such that they can form complete seals across the channels they are meant to cover. I generate a routing grid that represents the minimum spacing needed to prevent features from violating these layout constraints and use this grid to route the channels connecting the valves and ports on the control layer.



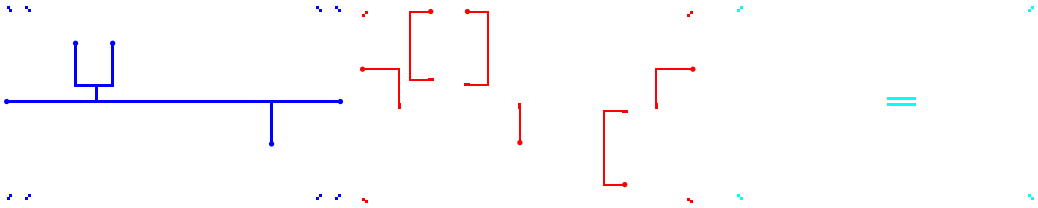
(a) Graph representation and flow layer layout of sample chip



(b) Placement of control valves

(c) Placement of control ports

(d) Routing of control layer



(e) Photomasks for flow layer, control layer and cell trap layer

Figure 3·6: Figure 3·6a shows the graph abstraction of the sample chip. Figures 3·6a through 3·6d show the physical design process from laying out the flow layer through routing of the control layer. The main features of the primary unit of the chip architecture are the 'stage' as shown in Figure 3·6a. The flow layer (3·6a) is laid out based on the connections present in the flow layer graph, and valves (3·6b) and ports (3·6c) in the control layer are placed to minimize routing problems. Channels on the control layer are routed with the negotiated congestion algorithm. Figure 3·6e shows the photomasks generated for the sample chip for the flow, control, and cell trap layers.

Flow layer layout

I convert the graph of gates to the graph representation of the flow layer going from the signal outputs to signal inputs. The signal flow will be from left to right, and the

media and cell sample flow will be from top to bottom. I create a sink port in the flow layer for each output and then expand leftward to create the logic stages. For each stage, I create and connect the set of cell traps, and connect the cell traps to their associated ports and the signal input and output nodes for the stage. The input node for one stage becomes the output node for the stage that precedes it. This continues until I reach the signal inputs. Here I generate a bank of cell traps for the biological sensors meant to convert the signal of interest into the intercellular signaling chemical and the cell sample and media source ports for those cell traps. Input source ports and output sink ports are arranged vertically while the media and cell sample source ports are arranged horizontally. The nodes in the flow layer are assigned preliminary coordinates that comply with the layout constraints, which are then converted to the nearest routing grid coordinates. The flow layer of a single stage is shown in Figure 3·6a. I can convert the graph of gates to the graph of the flow layer in $O(n)$ time where n is the number of gates needed in the original logic function.

Control valve and port placement

In general, valves on flow channels are placed near the target node in the direction of fluid flow. For flow channels that begin or terminate at a flow port, I place the valve at least the minimum allowed distance away from the port. For multiplexers, I stagger the valve placement such that valves assigned to the same control line are arranged in a horizontal or vertical line depending on the flow direction of the binary tree flow channel structure. The valve placement for the sample chip is shown in Figure 3·6b.

Control ports are placed to maximized routablility of the control layer. I split control port placement for multiplexers so that each line of valves has unimpeded routing access to a control port. Control ports for multiplexers containing flow ports are placed flanking those flow ports, with half the control ports on each side. For

cell trap multiplexers, I align half the control ports above the cell traps and half to the below them. For single valves controlling a single flow port, I place the control port for that valve a minimum distance below the flow port. Control ports for single valves with no other associated structure are placed the minimum safe distance below the valve. The control port placement for the sample chip is shown in Figure 3·6c.

Control layer routing

The routing algorithm is based on the negotiated congestion routing (Ebeling et al., 1995; Betz and Rose, 1997). Each location on the routing grid where a control channel may be placed is a routing resource. As the control layer is only a single layer, the channels may not cross, and each routing resource may only be used once. The algorithm takes as an input the control layer graph and produces the list of routing resources used for each path. Each edge in the control layer graph represents a signal that must be routed. Control channels must start and end at a valve or control port, and may not otherwise intersect those features. While control channels may cross flow channels, they should only do when there is no other path, and no control channel should directly overlay a flow channel. The width of a control channel is reduced when it crosses a flow channel to lessen the chance that it would obstruct the flow. Control channels are not allowed within the minimum safe distance of any control or flow port other than the one they connect to.

There are two phases to the routing, the global routing and the individual signal routing. The signal router uses A* search to find the best path between the source and the target of the signal being routed while the global router handles the allocation of shared routing resources. In the signal router, I assess penalties for the path crossing flow channels and for having multiple bends as channels with few bends are easier to fabricate. In the first pass of the global router, paths are routed without regard to shared resources so that more than one path can use the same resource. Once all the

```
1: while shared resources exist do
2:    for each edge e in the control layer graph do
3:       Find start and end routing resources
4:       if path p exists between start and end resources and path p contains a shared
         resource then
5:          Clear contents of path p
6:       end if
7:       Find path p from start resource to end resource with A* search
8:       for resource n in path p do
9:          Update penalty p_n and resource cost c_n
10:      end for
11:   end for
12:   check for shared resources
13:   Update historical penalty h_n for all nodes
14: end while
```

Figure 3·7: Negotiated Congestion Routing, $c_n = (b_n + h_n) * p_n$ where b_n is the base cost, h_n is the history of congestion on n in previous cycles, and p_n is related to the number of other paths using routing resource n currently.

paths have been routed, I check which paths use a resource that is used by another path. All such paths that share resources are then removed. The cost of using any routing resource that was shared on the previous iteration is then increased, and the paths that have been removed are then rerouted. This continues until there are no more shared resources. The pseudocode for this algorithm is shown in Figure 3·7, and the routed control layer for the sample chip is shown in Figure 3·6d.

Microfluidic valve control generation

I generate Java files to interface to the existing control software BioStream, which was developed to design GUIs and control valves for multilayer devices (Thies et al., 2008; Amin et al., 2009; Urbanski et al., 2006) (freely available at (Thies et al., 2009)). These files map each control line to one of the 48 physical ports on the control apparatus and define the mechanisms for opening and closing a port and for closing or opening all the ports on the chip.

I generate control patterns by first defining the default state of the chip, when it

has media flow through all cell traps to the waste ports with each stage operating independently from other stages. Valves allowing such behavior are set to the `OPEN` state, and all other valves are set to the `CLOSED` state. For all other control patterns, any valve that does not have its state set will be in the default state. I next define a set of flow paths for four possible operations in a single flow stage: loading cells and media to each cell trap individually, flushing media through all cell traps simultaneously, flushing the contents of each cell trap to the output node, and applying the inputs to each appropriate cell trap.

The number of total operations defined is based on the number of cell traps present in the flow layer. I estimate this as the number of nodes in the flow layer in the worst case. For each operation, I find the path through the control layer graph from the source port or input node to the waste port or output node for each cell trap. Each edge in the path is a channel that must remain open. If that channel has a valve placed on it, then the valve's state is set to `OPEN`. Path searches can be performed in $O(n)$ time with breadth-first search, where n is the number of nodes in the flow layer graph. The total time to find all flow paths is then $O(n^2)$. I convert these flow paths to valve instructions in BioStream and output those instructions.

Photomask generation

I create photomasks for manufacturing of the chip based on the layout parameters and the flow and control layer graphs. Photomasks are used to created the master mold for the flow layer and control layer in multilayer soft lithography (Duffy et al., 1998; Sia and Whitesides, 2003). For each channel in the flow layer and control layer, I draw the channel based on the coordinates of its source and target nodes. The cell traps are drawn on a different layer than the flow channels as the cell traps will be a different depth than the channels. For the sample chip I described, I generate the three photomasks shown in Figure 3·6e, as well as reference image of the three layers

overlaid on each other. The reference image is used as the base for the chip level simulation.

3.3.3 Results

I tested the functionality of Fluigi with two sets of benchmarks. I first constructed all the possible two and three-input Boolean logic benchmark functions using exclusively `NOR` gates. Those with 0 inputs were ignored as those are not useful for demonstrating the capabilities of Fluigi. The results of Fluigi on the 14 two-input Boolean functions and 254 three-input Boolean functions using only `NOR` gates are shown in Table 3.6.

The second set of benchmarks demonstrated the generalized functionality of Fluigi on more complex functions. I also tested Fluigi on four specific example circuits: 1) a 4 input `AND` gate, the genetic circuit of which is described in (Moon et al., 2012), 2) a 3 input `XOR` gate, as a representative of the three-input Boolean logic function benchmarks, 3) the half adder, a simpler version of the genetic circuit described in (Beal et al., 2011), and 4) an 8 to 3 encoder, the genetic circuit of which is being developed in the Densmore lab. In addition, I adapted Fluigi to describe two non-combinatorial functions, the first a microfluidic chip for constructing the repressilator (Elowitz and Leibler, 2000) using three stages and a feedback channel and the second a three stage assay similar to a neural net.

3.3.4 Benchmark circuits

Under these constraints, all 14 two-input Boolean functions could be constructed using 5 stages of logic, counting the stage of input buffers used to translate the input signal to an intercellular communications signal, the two most complicated ones being the 2 input `XOR` and `NXOR` functions. These circuits were too small for multiplexing to reduce the number of control lines.

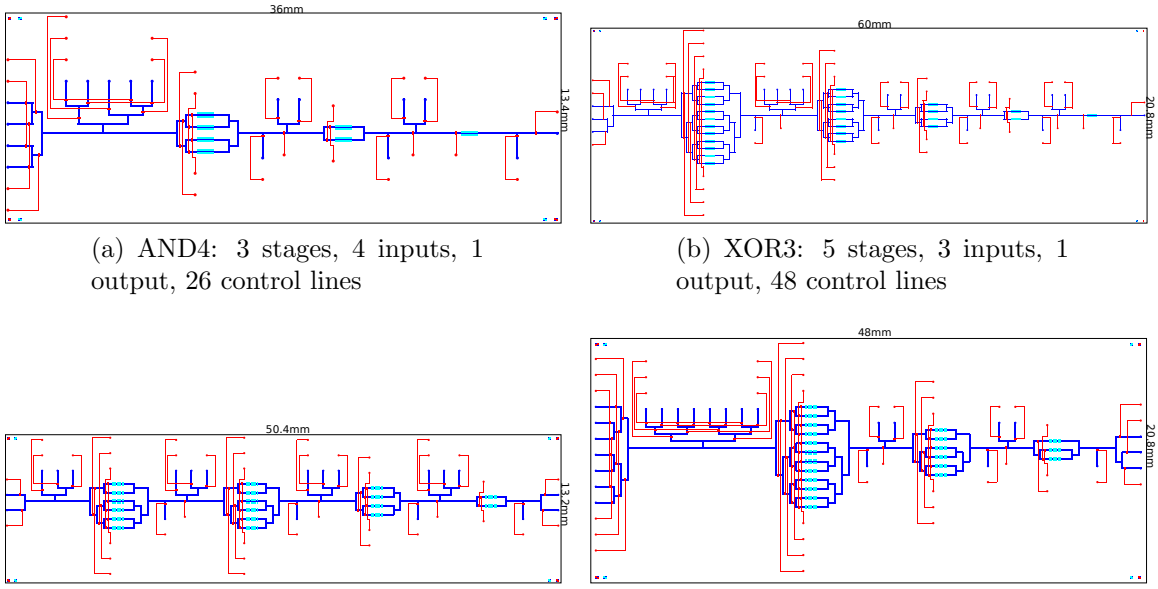
For three-input Boolean logic functions constructed with only `NOR` gates, the most

Table 3.6: Three-input Boolean logic functions

Number of Stages	Number of Circuits	Average Unoptimized Control Lines	Average Optimized Control Lines	Average Control Line Reduction	% of Circuits Compatible with Control Setup
Built with NOR Gates					
1	3	6	6	0.00%	100
2	6	12.50	12.50	0.00%	100
3	12	20.25	20.25	0.00%	100
4	22	27.73	27.73	0.00%	100
5	74	37.45	37.45	0.00%	100
6	26	46.65	46.65	0.00%	100
7	69	58.14	58.14	0.00%	0
8	20	71.75	69.55	3.02%	0
9	22	85.95	81.95	4.54%	0
Built with AND, OR, and NOT Gates					
1	3	6	6	0.00%	100
2	15	13.4	13.4	0.00%	100
3	53	23.37	23.37	0.00%	100
4	109	34.18	34.18	0.00%	100
5	75	49.32	48.21	1.97%	61.3

complicated functions took 9 stages of logic, including input buffers, to carry out. Only the largest circuits with 8 or 9 stages showed any benefit of control line optimization. With these three-input functions, I also notice that chips containing functions of 7 stages or more require more than 48 control lines to fully operate. Of the 254 benchmark functions, only 143 can be controlled by my existing hardware. This complication is due to the minimum number of control lines needed to operate a stage. The smallest functions requiring a single stage still require six control lines to operate, which limits me to a maximum of 8 stages of single gates. Increasing the number of gates in a stage will only increase the number of control lines used from the minimum.

To mitigate this problem, I ran all three-input benchmark circuits through Fluigi, and allowed the use of AND, OR, and NOT gates instead of relying exclusively on NOR gates. The new results for three-input benchmark functions are also shown in Table 3.6. Allowing the use of additional gates simplifies the circuits from a maximum of 9



(a) AND4: 3 stages, 4 inputs, 1 output, 26 control lines

(b) XOR3: 5 stages, 3 inputs, 1 output, 48 control lines

(c) Half Adder: 4 stages, 2 inputs, 2 outputs, 41 control lines

(d) 8-3 Encoder: 3 stages, 7 inputs, 3 outputs, 41 control lines

Figure 3·8: Layout of selected combinatorial logic circuits as generated by Fluigi. These may be used to generate the photomasks for chip fabrication. Blue represents the flow layer and red represents the control layer. Dimensions are provided for each design.

stages to a maximum of 5 stages. As previously, the reduction in control lines was only apparent with more complex functions. However, there still remained 29 functions that required over 48 control lines to operate. To reduce the number of control lines needed even further I would need to look at either reducing the minimum number of control lines needed in each stage, sharing control lines across stages, or better logic minimization to reduce the number of stages needed.

3.3.5 Example circuits

I selected four circuits, the four-input `AND` gate, the three-input `XOR` gate, the half adder, and the 8 to 3 encoder, to describe and examine in further detail. Based on my results with the three-input benchmark functions, I use a combination of `AND`, `OR` and `NOT` gates to construct these circuits to reduce the number of logic stages

Table 3.7: Example circuits

Circuit	Number of Stages	Average Unoptimized Control Lines	Average Optimized Control Lines	Average Control Line Reduction	Length (mm)	Width (mm)
Logic Functions						
AND4	3	26	26	0.00%	36.0	13.4
XOR3	5	54	48	11.11%	60.0	20.8
ADDER	4	41	41	0.00%	50.4	13.2
8-3 ENC3	3	48	41	14.58%	48.0	20.8
XOR4	7	97	77	20.62%	89.4	24.0
Non-combinatorial Logic Functions						
Oscillator	3	17	17	0.00%	29.4	8.8
Assay	3	49	46	6.12%	54.8	17.4

needed. For these four circuits, I compiled the valve control code Fluigi generated and successfully ran chip level behavior simulations of these circuits using existing microfluidics control software. The layouts for these four circuits as generated by Fluigi are shown in Figure 3·8, and the statistics for these are shown in Table 3.7.

Figures 3·8c and 3·8d show Fluigi's ability to layout and generate controls for logic functions with more than one output. The half adder is a simpler version of the genetic arithmetic circuits first described in (Beal et al., 2011) and is comprised of two smaller circuits, an `AND2` gate and an `XOR2` gate. Of the four sample circuits, only the three-input `XOR` gate and 8-to-3 decoder showed any benefit of multiplexing for control line optimization for relative optimization rates of 11.11% and 14.58% respectively. I also ran Fluigi on a 4-input `XOR` gate, which is a circuit larger than the biggest logic circuit constructed in synthetic biology thus far. The 4-input `XOR` gate is also large enough that multiplexing produces a 20.62% reduction in control line usage. However, it would still require 77 control lines to operate, and is outside the capabilities of my current hardware control system.

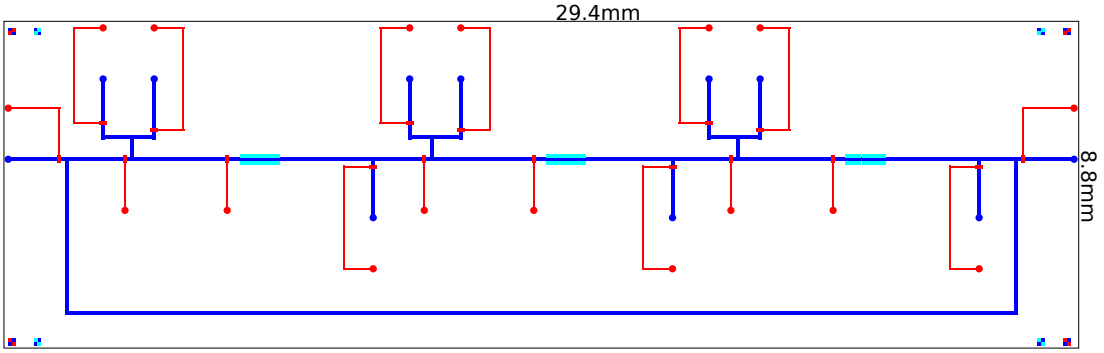
The benefits of multiplexing decrease as the number of cell trap blocks in each stage decreases, so multiplexing provides no benefit in later stages for circuits that

only have one output. With this in mind, I then created an alternate version of input specification for Fluigi to maximize the benefits of multiplexing. I constrain Fluigi to only creating three flow stages, but allow the user to specify the number of cell trap blocks per stage. The number of inputs into the system is the number of cell trap blocks in the first stage, and the number of outputs of the system is the number of cell trap blocks in the last stage. I assume for now that each cell trap block contains a different cell strain. With this input specification type, I also allow feedback from later stages to earlier stages. I created two functions, a three-stage oscillator shown in Figure 3·9a and a 3-stage assay shown in Figure 3·9b, with this alternate input specification to demonstrate that Fluigi is not limited to laying out logic functions. The results for these two chips are also shown in Table 3.3.

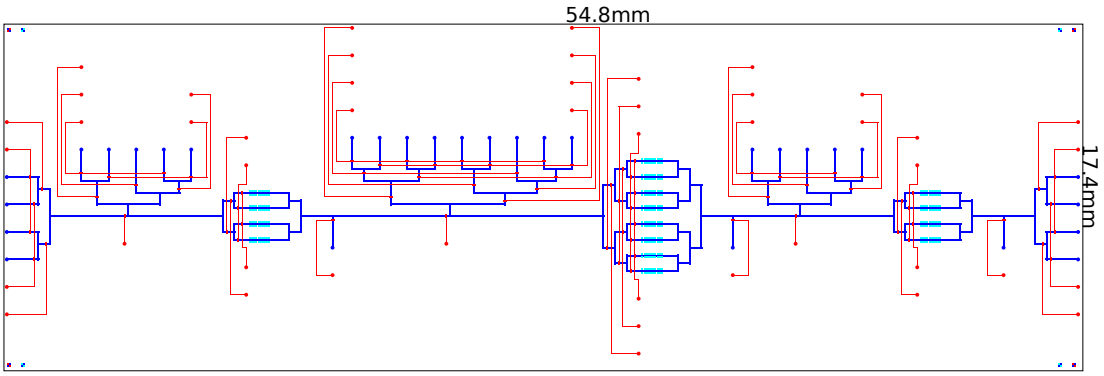
The oscillator demonstrates that my software is capable of creating feedback loops in the flow channel. In this system, each cell trap block will be loaded with a cell with a sensor for some input chemical that produces both a fluorescent protein and a chemical to inhibit the sensor in the cell in the following stage. The chip for this system has 17 control lines, with one control line per valve. As each stage already has the minimum possible number of control lines, no further optimization can be performed. The assay design is similar to designs of neural networks, with an input layer, a hidden/processing layer, and an output layer. Since this design has multiple cell trap blocks in each layer, multiplexing allows savings of 6.12% in control line usage.

3.3.6 Flaws

The biggest flaws in this iteration of the workflow were the limited application of the proposed device design and the reliance on heuristics when generating a device design. Ideally, the workflow would apply to a wider range of problems than simply testing biological Boolean logic. To do so, I would need more flexibility in the architecture for



(a) 3 stage oscillator based on the repressilator



(b) 3 stage assay of 4 samples, 8 samples and 4 samples

Figure 3·9: Layout of selected non-combinatorial circuits as generated by Fluigi. These images serve as the GUI for simulation of high level chip behavior and may be used to generate the photomask for chip fabrication. The sections in blue represent the flow layer and the sections in red represent the control layer. The chip dimensions are provided for each chip.

describing microfluidic structures rather than the limited set used for implementing testbeds for biological Boolean logic. Ideally, the change in architecture would allow me to move away from using heuristically generated layouts.

Chapter 4

Design architecture

In this chapter, I discuss the model I use to represent a microfluidic device, the implementation of the model, and the rules of the netlist file format used to describe microfluidic devices. Sample netlists for microfluidic devices are included in Appendix B.

4.1 Device model

I represent a microfluidic device d by a graph $G(V, E)$. Each vertex $v \in V$ represents a microfluidic component such as but not limited to an input/output port, a valve, a serpentine mixer, a cell trap, or another basic structure commonly found in microfluidic devices. Each edge $e \in E$ represents a microfluidic channel connecting two components. For each vertex $v \in V$, $\exists$ a set L of points $p = p_x, p_y$ that represent coordinates of the input/output terminals of the component. The degree of vertex $v_i \in V$ may not be greater than the size of set L_i. For each degree of v_i, $\exists$ channel $e \in E$ that has its source or target at those coordinates.

The flow layer and control layer can be represented by the subgraphs $F \in G$ and $C \in G$ respectively. For more complex but commonly used microfluidic structures such as multiplexers or rotary pumps with components on multiple layers, I define a module m as a subgraph of G that can contain elements from both F and C. I define the set of modules $m_1 \dots m_n \in G$ as M. I represent a set of channels that must be routed together as as a net $n \subset E$, and N as the set of nets $n_1 \dots n_n \in G$.

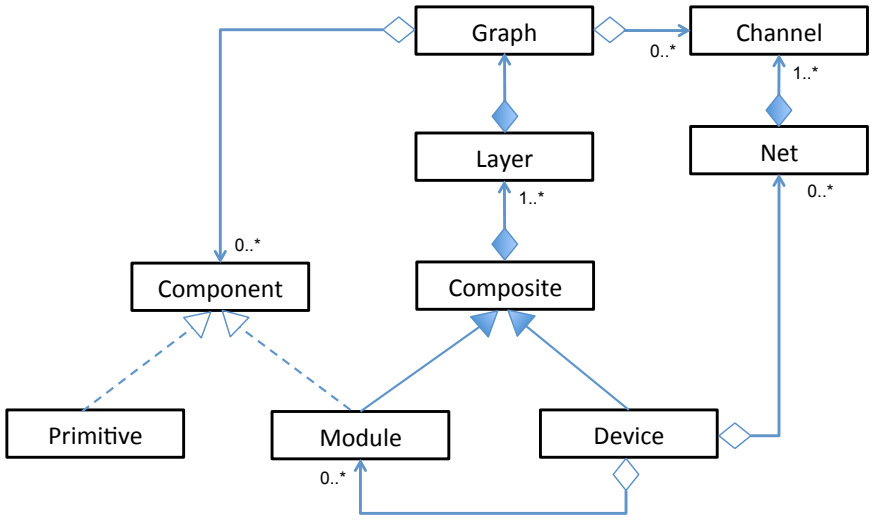


Figure 4·1: Class diagram of architecture model. A microfluidic device is represented as a `Device` object which contains multiple `Layer` objects, and the features and connections on each `Layer` are represented by a `Graph` of `Component` objects as vertices and `Channel` objects as edges. The `Component` interface is implemented by two classes, `Primitive` and `Module`. A `Primitive` represents a basic feature of a microfluidic device, and a `Module` is a collection of multiple `Primitive` objects that perform a more complex function. Dashed lines represent interfaces to be implemented. Solid arrows represent subclasses. Diamonds represent collections.

4.2 Implementation

I implemented the device model in Java using the class diagram shown in Figure 4·1. The most important piece in the diagram is the `Component` interface, which represents the physical characteristics of a microfluidic structure, including its location on a 2-D plane with the origin at the upper left, the minimum boundary space around it, and the number and location of its terminals. The x, y location of all components is referenced by the coordinates of the upper left corner of the component. Two classes, `Primitive` and `Module`, implement this interface. Here, I consider a `Primitive` object to describe the most basic microfluidic structure that can be used to build larger structures. `Primitive` objects in my architecture include ports, cell traps, valves, serpentine mixers, and nodes, which are locations where channels may intersect.

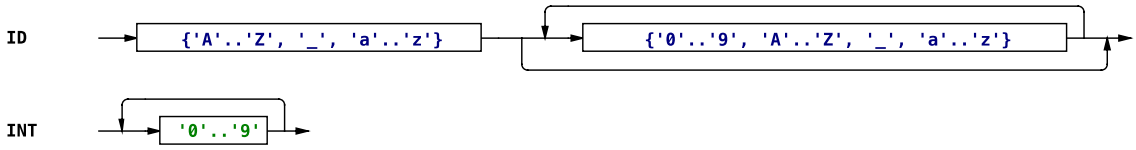


Figure 4·2: Lexical rules for the netlist grammar. The lexer has two rules, `ID` and `INT`. An `ID` may be any combination of upper and lower case letters and numbers, but must start with an upper or lower case letter. An `INT` may be any combination of numbers, and does not include decimal points. The lexer ignores white space.

I introduce the `Composite` class for describing more complex microfluidic structures that may span multiple layers. A `Composite` contains one or more `Layer` objects, and each `Layer` contains a `Graph` of `Component` objects as vertices and `Channel` objects as edges. Both the `Module` class and the `Device` class are subclasses of `Composite`. As `Module` implements the `Component` interface, it is possible for a `Device` or `Module` to contain other `Module` objects.

The `Channel` class represents a physical connection between two `Component`s, and exists as an edge in a `Graph`. In addition to its source and target in the graph, a `Channel` also contains the source terminal and target terminal it connects at and their respective coordinates. The `Net` class represents a set of channels with the same source component and the same source terminals. All `Channel`s in a `Net` are treated as one object for the purposes of the place-and-route tool, and are routed at the same time.

4.3 Netlist rules

In this section I give the detailed syntax of my microfluidic netlist format, and the grammar rules for it. I used `ANTLR3.5.2` to write my grammar and associated lexer and parser. I have two rules for the lexer, the definition of `ID` as component names and the definition of `INT` as numbers, as shown in Figure 4·2. An `ID` may be any combination of upper and lower case letters and numbers, but must start with an

```
DEVICE test01

LAYER FLOW
PORT p1, p2, p3 r=500;
NODE n1;
H LONG CELL TRAP ct1 numChambers=10 chamberWidth=500
chamberLength=500 chamberSpacing=100 channelWidth=500;

CHANNEL c1 from p1 3 to n1 1 w=500;
CHANNEL c2 from p2 1 to n1 3 w=500;
CHANNEL c3 from n1 2 to ct1 1 w=500;
CHANNEL c4 from ct1 2 to p3 4 w=500;
END LAYER

LAYER CONTROL
PORT cp1, cp2 r=500;
VALVE v1 on c1 w=1000 l=500;
VALVE v2 on c2 w=1000 l=500;
CHANNEL c5 from cp1 2 to v1 4 w=500;
CHANNEL c6 from cp2 2 to v2 4 w=500;

END LAYER
```

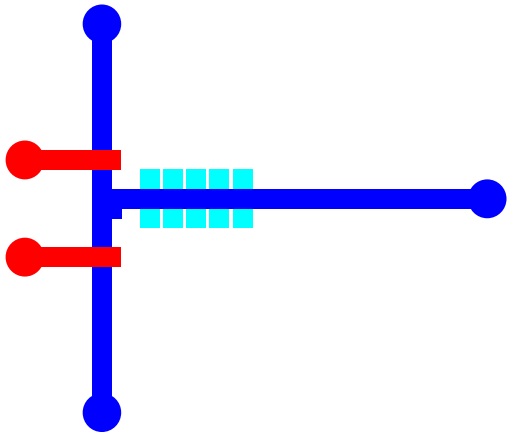


Figure 4·3: Netlist example. The netlist describes a simple device with three ports and a cell trap on the flow layer and two valves and ports on the control layer. The name of the device is given in the first line. The features and connections on the flow layer are described first, followed by those on the control layer. The design described in the netlist is shown on the right.

upper or lower case letter. An `INT` may be any combination of numbers, but does not include decimal points. I assume that all numerical parameters are in microns. White space is ignored by the lexer.

4.3.1 Device declaration

A simple microfluidic device and netlist for describing it are shown in Figure 4·3. The basic structure of the netlist as follows:

1. Declare the type of device and the device name.
2. Declare each layer present in the device.
3. Within each layer declare the components and channels on that layer.
4. Within each layer, set the position of the components in that layer.

Figure 4·4A and B shows the parser rules for the structure of the netlist. The first line of the netlist file is the header statement that includes the type of device

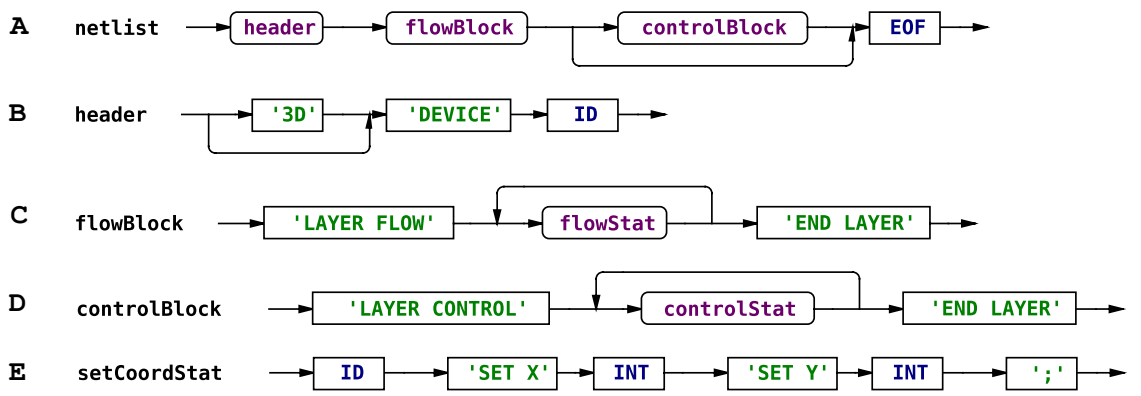


Figure 4·4: Syntax for netlist grammar. The netlist is broken up into the `header`, the `flowBlock`, and the optional `controlBlock`, and ends with a `newline`. The `header` contains the device name, and the device fabrication method in optional `3D` flag. The `flowBlock` contains the declarations of all the features on the flow layer, and the `controlBlock` contains the declarations of all the features on the control layer. The explicit position of a feature may be set by the `setCoordStat` statement inside the `flowBlock` or the `controlBlock`.

and the device name. Using the optional `3D` tag declares the netlist to describe a device meant to be manufactured by 3D printing or a computer numerical control (CNC) milling platform (Duncan et al., 2015). If this option is used, the device will be marked with the `3D` flag. This flag allows usage of the structures `3DVALVE`, `VIA`, and `TRANSPOSER` but block the usage of the structures `MUX`, `ROTARY MIXER`, and `LOGIC ARRAY` as those modules have not been adapted to use 3D Valves. See Section 4.3.5 for more information about 3D structures.

I limit the device to a maximum of two layers, a flow layer and a control layer. The control layer is optional, as there are many devices that only have a flow layer. The syntax for layer statements is shown in Figure 4·4C and D. Figure 4·5 shows the valid statements in each layer. All modules are declared on the flow layer, even those that span both layers. The position of a component on a layer may be set with the `setCoordStat` shown in Figure 4·4E. In this statement, `ID` is the name of the component, and the component is moved so its upper left corner is at the position

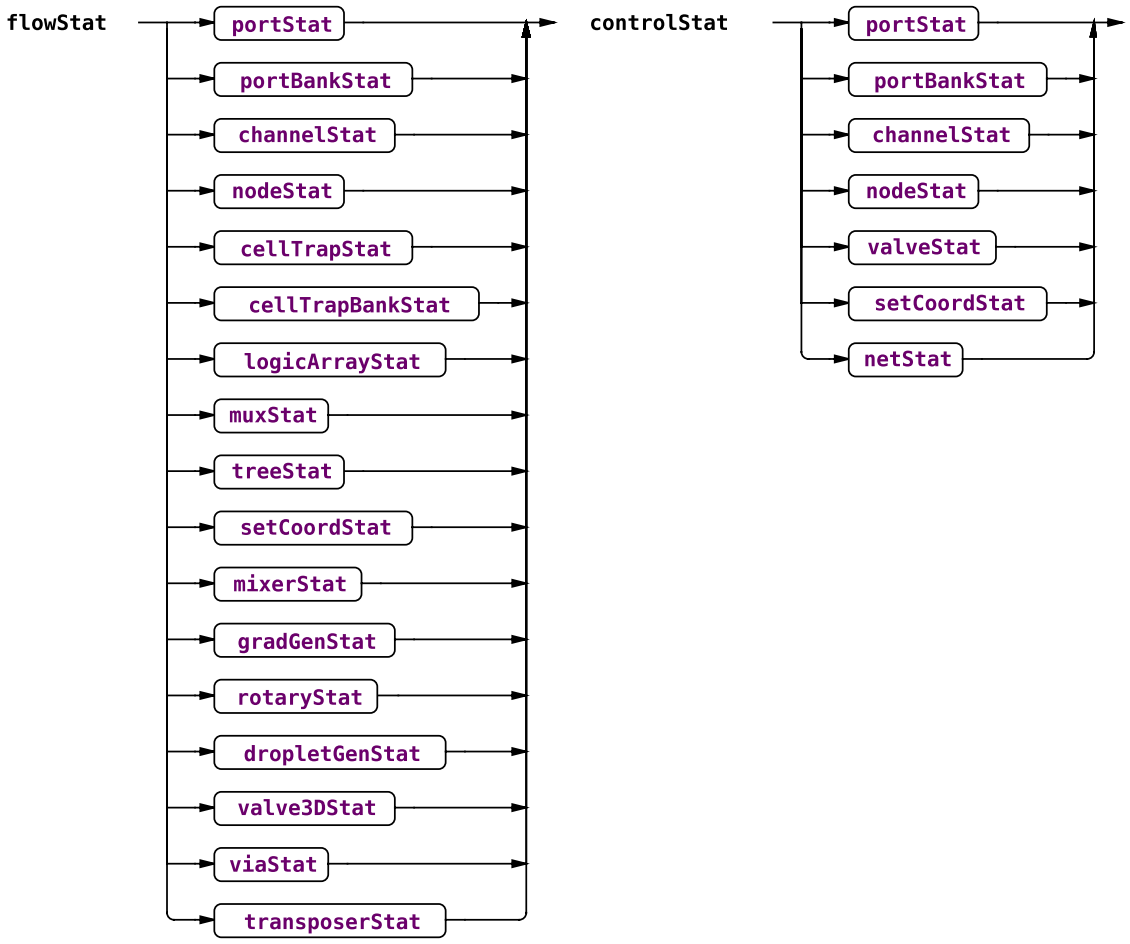


Figure 4·5: Syntax for layer declarations. Cell traps, mixers, and modules with features on both flow and control layers may only be declared on the flow layer. Valves and nets may only be declared on the control layer. Ports, port banks, nodes, and channels may be declared on any layer.

referenced.

4.3.2 Primitives

I show in Table 4.1 the syntax for declaring each type of `Primitive`. Subclasses of `Primitive` include `Port`, `Node`, `CellTrap`, `Mixer`, and `Valve`.

Table 4.1: Syntax for statements declaring primitives

Primitive	**Syntax**
Port	'PORT' `ID` (',' `ID`)* ('r=' `INT`)? ';'
Node	'NODE' `ID` (',' `ID`)* ';'
Long Cell Trap	('V' |'H') 'LONG CELL TRAP' `ID` (',' `ID`)* 'numChambers=' `INT` 'chamberWidth=' `INT` 'chamberLength=' `INT` 'chamberSpacing=' `INT` 'channelWidth=' `INT` ';'
Square Cell Trap	'SQUARE CELL TRAP' `ID` (',' `ID`)* 'chamberWidth=' `INT` 'chamberLength=' `INT` 'chamberSpacing=' `INT` 'channelWidth=' `INT` ';'
Mixer	('V' |'H') 'MIXER' `ID` 'numBends=' `INT` 'bendSpacing=' `INT` 'bendLength=' `INT` 'channelWidth=' `INT` ';'
Valve	'VALVE' `ID` 'on' `ID` 'w=' `INT` 'l=' `INT` ';'

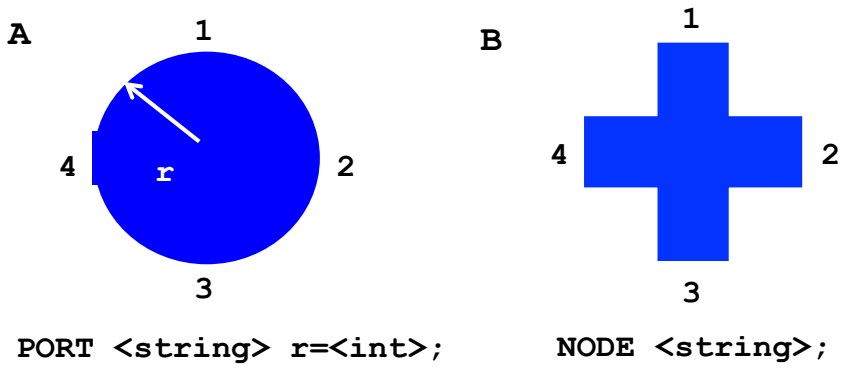


Figure 4·6: The diagram of parameters and terminal numbering for `Port` is shown in A. A `Port` represents a location on the device where fluid or air may enter and leave and has a radius defined in microns. The diagram of parameters and terminal numbering for `Node` is shown in B. A `Node` represents a location on the device where up to four channels on the same layer may intersect.

Port

A `Port` represents a location on the device where fluid or air may enter and leave. `Port` objects may be declared on both the flow and control layer. It has one parameter, the radius, which must be given in microns. A diagram of the parameters and numbering and location of the terminals is shown in Figure 4·6A. The syntax is shown in Table 4.1. Multiple ports may be declared in the same statement provided they have the same radius.

Node

A `Node` represents a location on the device where up to four channels on the same layer may intersect. It has no parameters, and its size is determined by the widths of the channels connecting to it. A diagram of the numbering and locations of the terminals is shown in Figure 4·6B. The syntax is shown in Table 4.1. Multiple nodes may be declared in the statement.

Cell trap

A `CellTrap` represents a structure for growing cells for microscopy. I based this structure on the cell trap design from the Hasty Lab (Prindle et al., 2012; Danino et al., 2010). It has multiple small chambers flanking a wider and deeper channel, all of which is treated as one structure. A separate photomask is generated for the chambers of the cell trap as they are a different height than the flow channels.

Figure 4·7 and Figure 4·8 show the parameters and terminal numbering for the two cell trap subclasses. Key parameters for a `CellTrap` are the length and width of chambers, distance between chambers, width of the flow channel, and the total number of chambers. A `CellTrap` may only be declared on the flow layer. The syntax for this statement is shown in Table 4.1. Multiple cell traps may be declared in the same statement provided they are of the same subclass and have the same parameters.

A long cell trap, `CellTrapL`, may be horizontal or vertical and has two terminals numbered as shown in Figure 4·7. It may contain an arbitrary number of chambers.

A square cell trap, `CellTrapS`, contains 4 chambers at a 4 way junction. It has 4 terminals numbered as shown in figure Figure 4·8. The size of chambers may vary, but this structure is restricted to 4 chambers only.

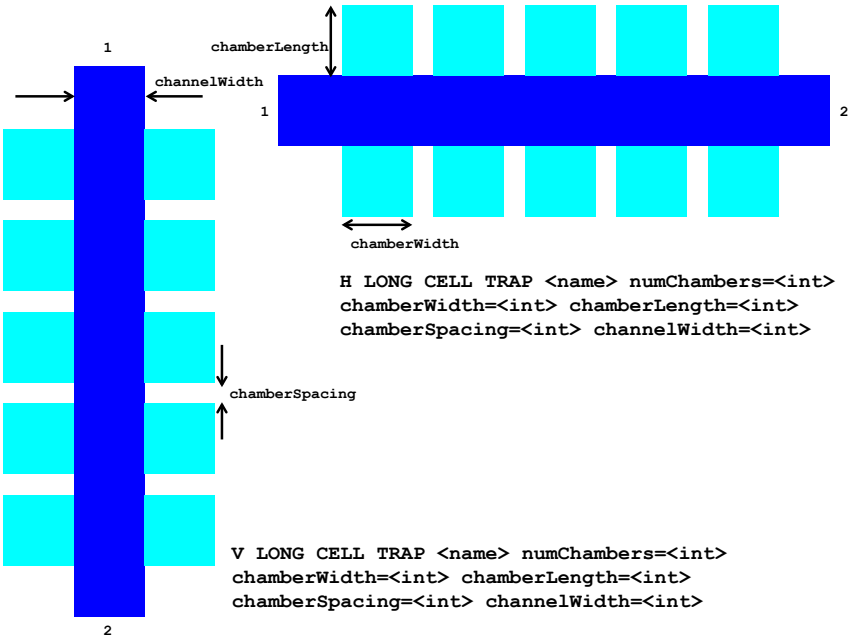


Figure 4·7: Diagram of parameters and terminal numbering for `CellTrapL`. The long cell trap has chambers for cell growth flanking a wider and deeper channel. The size of the chambers and the spacing between the chambers affects the performance of the cell trap.

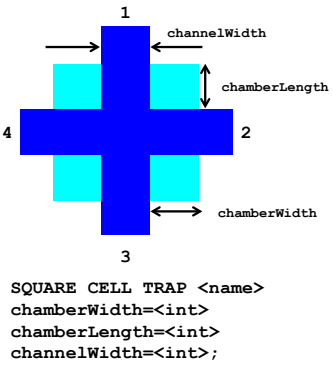


Figure 4·8: Diagram of parameters and terminal numbering for `CellTrapS`. The square cell trap contains 4 chambers at a 4 way junction of channels.

Mixer

A `Mixer` represents a serpentine mixer used for mixing two fluids. The details of the fluid mechanics of this structure are given by Squires and Quake (Squires and

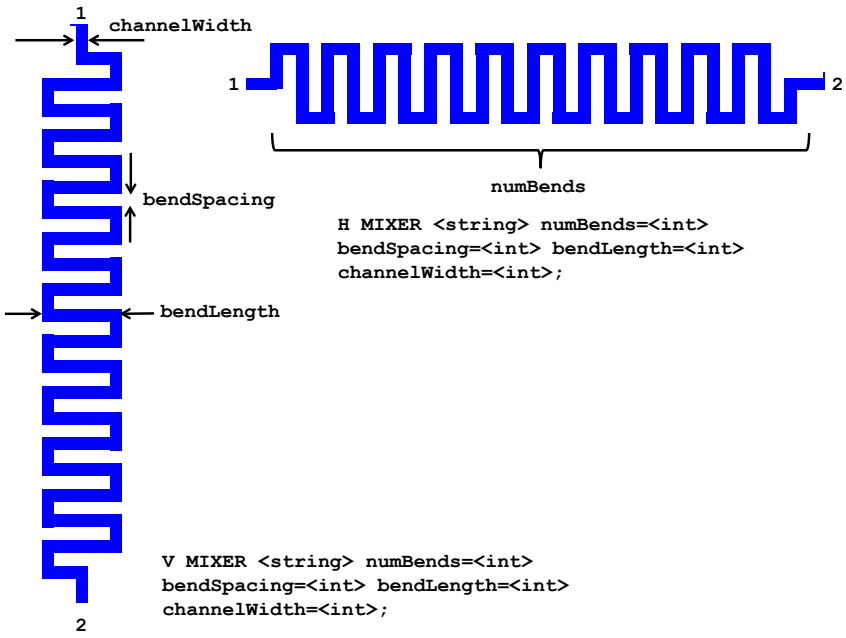


Figure 4·9: Diagram of parameters and terminal numbering for `Mixer`. A `Mixer` represents a serpentine mixer used for mixing two fluids. The details of the fluid mechanics of this structure are given by Squires and Quake (Squires and Quake, 2005).

Quake, 2005). The key parameters are the number of bends, the spacing distance between bends, the length of bend, and the width of the channel. This structure may be horizontal or vertical and has two terminals. It may only be declared in the flow layer. The parameters and terminal numbering are diagrammed in Figure 4·9. The syntax for this statement is shown in Table 4.1.

Valve

A `Valve` represents a location on the control layer that may be distended through application of pneumatic pressure to form a seal on the flow layer. The full mechanics of this is described by Thorsen et al. (Thorsen et al., 2002). This may only be declared on the control layer. The parameters for a valve are the length, the width, and the flow channel the valve controls. The channel must exist on the flow layer in order for

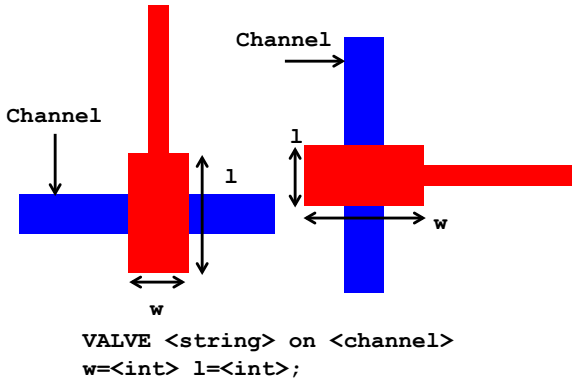


Figure 4·10: Diagram of parameters and terminal numbering for `Valve`. A `Valve` represents a location on the control layer that may be distended through application of pneumatic pressure to form a seal on the flow layer. The full mechanics of this is described by Thorsen and Quake (Thorsen et al., 2002).

the valve to be created. For a horizontal flow channel, the valve width is 1.5 times channel width, and the valve length is 3 times channel width. For a vertical flow channel, the valve width is 3 times channel width, and the valve length is 1.5 times channel width. The parameters and terminal numbering are shown in Figure 4·10. The syntax for this statement is shown in in Table 4.1.

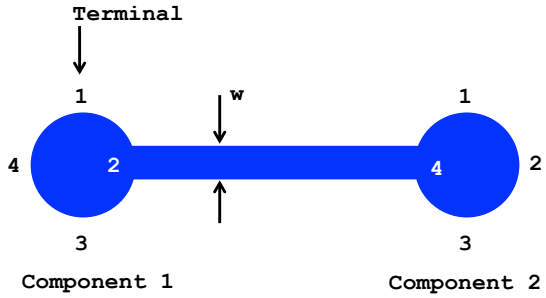
As an application note, if the valve is placed on a horizontal channel, control layer connections should not be made to terminals 1 and 3. Likewise, if the valve is placed on a vertical flow channel, control layer connections should not made to terminals 2 and 4. Doing so would cause a control channel to be physically on top of a flow channel, which would violate design rules for multilayer soft lithography. This structure is valid for valves made with multilayer soft lithography. Section 4.3.5 covers valve designs for alternative fabrication methods.

4.3.3 Channel

A `Channel` represents a connection between two structures on a microfluidic device. A channel goes from a terminal on a source component to a terminal on a target component. The key parameter of a channel is the width and is diagrammed in

Table 4.2: Syntax for statements declaring channels and nets

Statement	Syntax
Channel	'CHANNEL' `ID` 'from' `ID INT` 'to' `ID INT` 'w=' `INT` ';'
Net	'NET' `ID` 'from' `ID INT` 'to' `ID INT` (',' `ID INT`)+ 'channelWidth=' `INT` ';'




```
CHANNEL <string> from <Component 1> <terminal>
to <Component 2> <terminal> w=INT;
```

Figure 4·11: Diagram of parameters and terminal numbering for `Channel`. A `Channel` represents a connection between two structures on a microfluidic device. A channel goes from a terminal on a source component to a terminal on a target component.

Figure 4·11. The syntax for declaring a `Channel` is shown in Table 4.2. Both the source and target components must have already been declared in the device, and both components must have a valid terminal. Channels only go between two terminals and may only connect at 90 degree angles.

Net

A `Net` represents a set of channels originating from a single source. All channels in a net share the same source component and the same source terminal. A `Net` may only be declared on the control layer at this time. Figure 4·12 shows the usage of the net statement. The syntax is shown in Table 4.2. When declaring a net, the target closest to the source must be first in the list of targets.

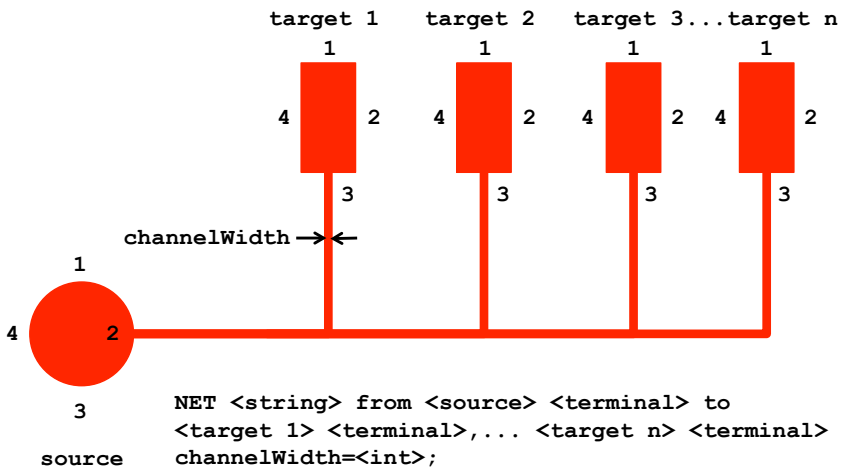


Figure 4·12: Diagram of parameters and terminal numbering for `Net`. A `Net` represents a set of channels originating from a single source. All channels in a net share the same source component and the same source terminal.

4.3.4 Modules

A `Module` represents commonly used complex structures found in microfluidic devices and contains elements on both the flow layer and the control layer. Components and channels contained in modules do not undergo the place-and-route as the entire module is treated as one component for the purposes of place-and-route. This allows modules to contain some structures involving channels at angles that would otherwise violate my architecture model. Table 4.3 shows the syntax for all module declarations. All modules with elements in the flow layer must be declared in the flow layer. The only module that can be declared in the control layer is a `PortBank`.

Bank

A `Bank` represents a set of repeated components with the same parameters placed a fixed distance from each other either vertically or horizontally. Key parameters include the number of components, the distance between terminals, and the width of the channels connecting to the components on each layer. Spacing contraints between

Table 4.3: Syntax for statements declaring modules

Statement	**Syntax**
Port Bank	'V' 'BANK' `ID` 'of' `INT` 'PORT' 'r=' `INT` 'dir=' ('RIGHT' |'LEFT') 'spacing=' `INT` 'channelWidth=' `INT` ';' 'H' 'BANK' `ID` 'of' `INT` 'PORT' 'r=' `INT` 'dir=' ('UP' |'DOWN') 'spacing=' `INT` 'channelWidth=' `INT` ';'
Cell Trap Bank	('V' |'H') 'BANK' `ID` 'of' `INT` 'CELL TRAP' 'numChambers=' `INT` 'chamberWidth=' `INT` 'chamberLength=' `INT` 'chamberSpacing=' `INT` 'spacing=' `INT` 'channelWidth=' `INT` ';'
Mux	('V' |'H') 'MUX' `ID` `INT` 'to' `INT` 'spacing=' `INT` 'flowChannelWidth=' `INT` 'controlChannelWidth=' `INT` ';'
Tree	('V' |'H') 'TREE' `ID` `INT` 'to' `INT` 'spacing=' `INT` 'flowChannelWidth=' `INT` ';'
Logic Array	'LOGIC ARRAY' `ID` 'flowChannelWidth=' `INT` 'controlChannelWidth=' `INT` 'chamberLength=' `INT` 'chamberWidth=' `INT` 'r=' `INT` ';'
Gradient Generator	('V' |'H') 'GRADIENT GENERATOR' `ID` `INT` 'to' `INT` 'numBends=' `INT` 'bendSpacing=' `INT` 'bendLength=' `INT` 'channelWidth=' `INT` ';'
T Droplet Generator	('V' |'H') 'DROPLET GENERATOR' 'T' `ID` 'radius=' `INT` 'oilChannelWidth=' `INT` 'waterChannelWidth=' `INT` ';'
Flow Focusing Droplet Generator	('V' |'H') 'DROPLET GENERATOR' 'FLOW FOCUS' `ID` 'radius=' `INT` 'oilChannelWidth=' `INT` 'waterChannelWidth=' `INT` 'angle=' `INT` 'length=' `INT` ';'
Rotary Pump	('V' |'H') 'ROTARY PUMP' `ID` 'radius=' `INT` 'flowChannelWidth=' `INT` 'controlChannelWidth=' `INT` ';'

adjacent components in each `Bank` are maintained. The two subclasses of `Bank` are `PortBank` for repeated ports and `CellTrapBank` for repeated long cell traps.

Additional parameters for `PortBank` are the radius of the ports in the bank and the direction that the connections are made from the ports. Figure 4·13 shows the parameters and terminal numbering. The syntax for declaring a port bank is shown in Table 4.3. `PortBank` is the only module that may be declared on both the flow layer and the control layer.

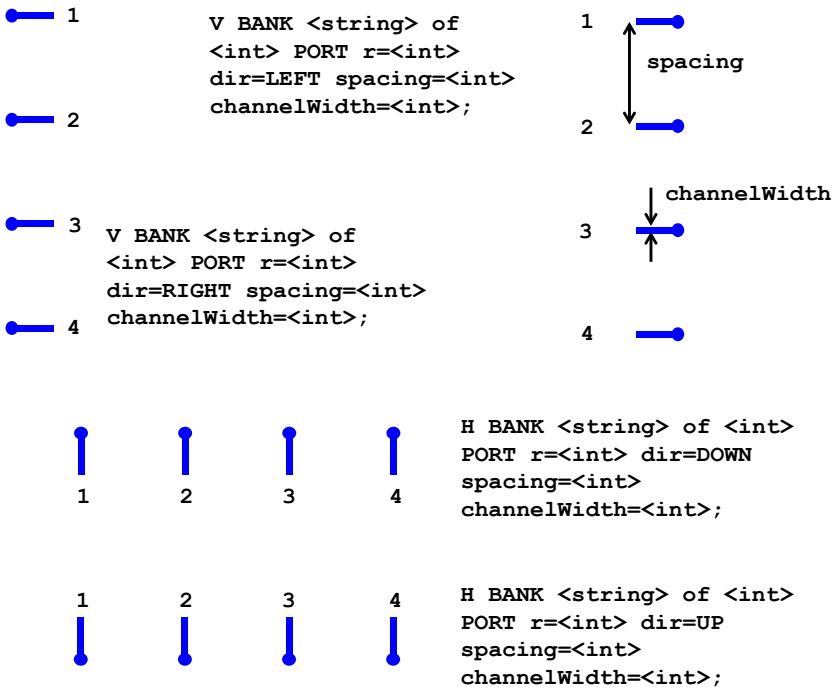


Figure 4·13: Diagram of parameters and terminal numbering for `PortBank`. A `PortBank` is a set of repeated ports with the same radius placed a fixed distance from each other.

Additional parameters for `CellTrapBank` are the parameters for defining the structure of each cell trap as seen in Figure 4·7, including number of chambers, the chamber size, and the spacing between chambers. All cell traps in a `CellTrapBank` have the same parameters. Figure 4·14 shows the terminal numbering, and the syntax for the statement is shown in Table 4.3.

Mux

A `Mux` represents the microfluidic multiplexer as described by Thorsen et al. (Thorsen et al., 2002). It uses $2\log_2 n$ control lines to select one of n flow channels. The parameters for declaring a `Mux` include number of flow lines to be selected from (1 to n or n to 1), the orientation (vertical or horizontal), the flow channel width, the control channel width, and the spacing between the terminals. Spacing between the

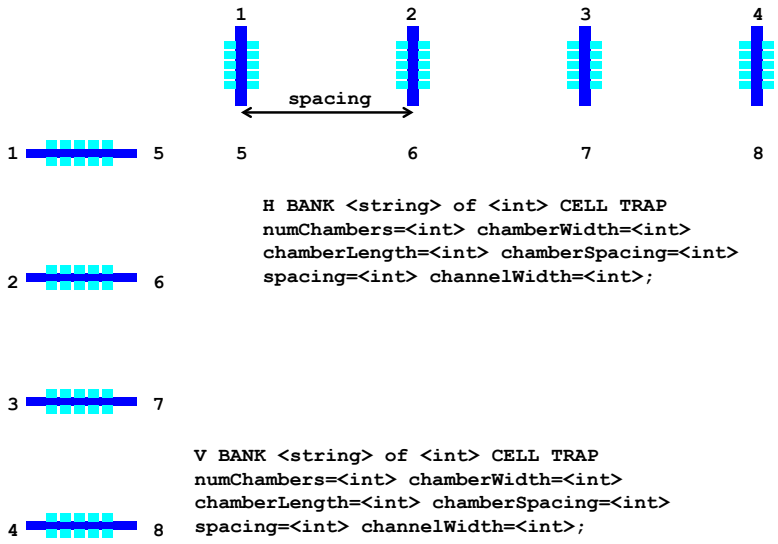
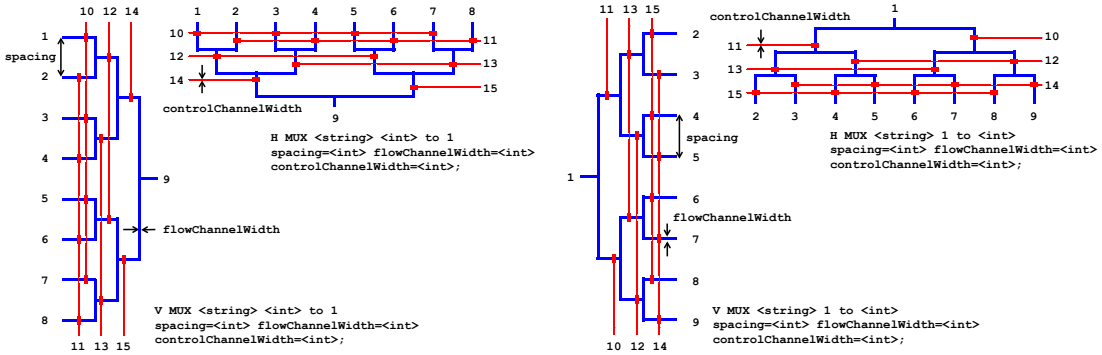


Figure 4·14: Diagram of parameters and terminal numbering for `CellTrapBank`. A `CellTrapBank` is a set of repeated cell traps with the same chamber size, chamber spacing, and channel width placed a fixed distance from each other.

channels is constrained to a minimum of the spacing required by the given design rule parameters (see Table 5.1. Valves are sized according to flow channel width, with the larger dimension being 3 times the flow channel width and the smaller dimension being 1.5 times the flow channel width. Figure 4·15 shows the parameters and terminal numbering for an 8 to 1 `Mux` and a 1 to 8 `Mux`. The syntax for this declaration is shown in Table 4.3.

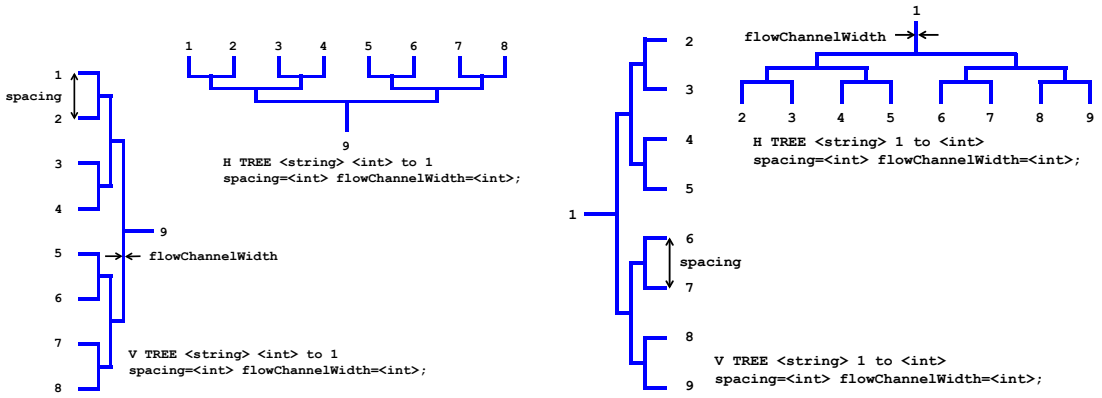
Tree

The structure of a `Tree` is that of a `Mux` without the control layer. A `Tree` has all the parameters of a `Mux` except the control channel width. Figure 4·16 shows the parameters and the terminal numbering, and the syntax is given in Table 4.3.



(a) n to 1 Multiplexers (b) 1 to n Multiplexers

Figure 4·15: Diagram of parameters and terminal numbering for `Mux`. A `Mux` represents the microfluidic multiplexer as described by Thorsen et al. (Thorsen et al., 2002). It uses $2\log_2 n$ control lines to select one of n flow channels.



(a) n to 1 Tree (b) 1 to n Tree

Figure 4·16: Diagram of parameters and terminal numbering for `Tree`. A `Tree` shares the same flow layer structure as a `Mux` and is used to combine multiple flows or split a single flow.

Logic array

The `LogicArray` is a structure for controlling flows between four square cell traps and an input. The cell traps are meant to contain biological Boolean logic. Each valve is controlled individually, and the structure can direct flow between any two or more cell traps. The key parameters include the length and width of the cell traps, the flow channel width, the control channel width, and the radius for the four internal

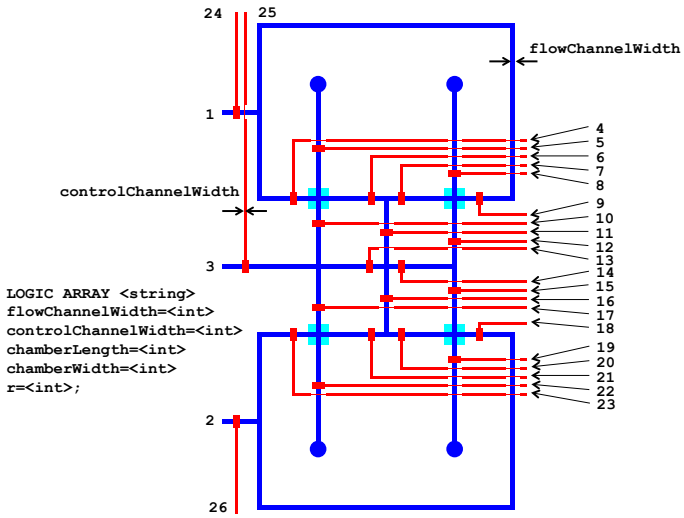


Figure 4·17: Diagram of parameters and terminal numbering for `LogicArray`. The `LogicArray` is a structure for controlling flows between four square cell traps and an input. The cell traps are meant to contain biological Boolean logic. Each valve is controlled individually, and the structure can direct flow between any two or more cell traps.

ports. The size of the valves will depend on the flow channel width. Parameters and terminal numbering are shown in Figure 4·17, and the syntax is given in Table 4.3.

Gradient Generator

The `GradientGenerator` is based on the design by Dertinger et al. (Dertinger et al., 2001). The arrangement of serpentine mixers generates a gradient of concentrations at the output channel. This is one of three modules that contain internal structures that would otherwise violate the architecture model. This structure contains a node that may have more than 4 connections with channels that do not connect at right angles.

Key parameters are the number of mixers at the input, the number of mixers at the output, and the channel width. The output channel width will be 3 times the channel width. Additional parameters used to define the structure of each mixer include bend length, number of bends, and the spacing between bends as shown in

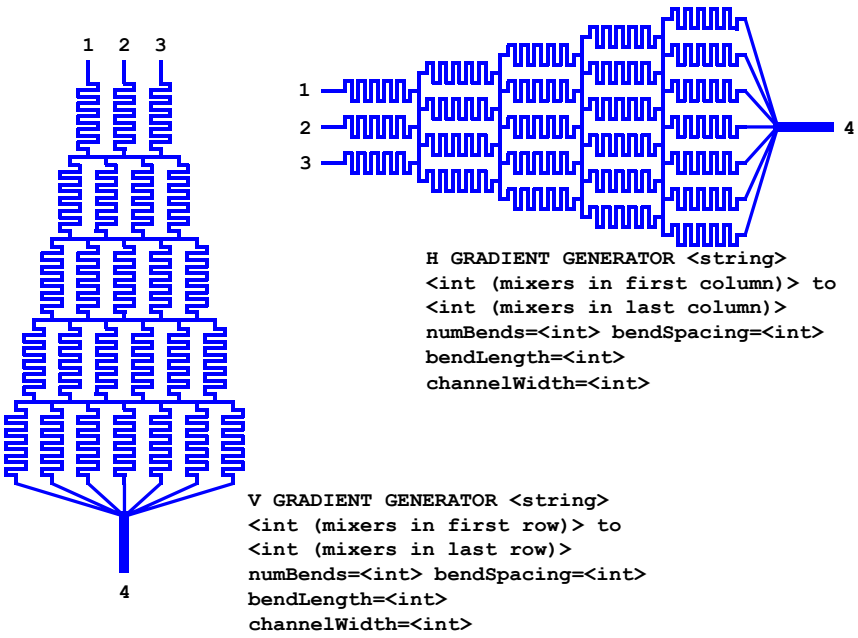


Figure 4·18: Diagram of parameters and terminal numbering for `GradientGenerator`. The `GradientGenerator` is based on the design by Dertinger et al. (Dertinger et al., 2001) and has an arrangement of serpentine mixers that generate a gradient of concentrations at the output channel.

Figure 4·9. Terminal numbering is shown in Figure 4·18, and the syntax is given in Table 4.3.

Droplet generator

Droplet generators are used to create droplets of water emulsified in oil for droplet or digital flow-based microfluidic experiments, the details of which are given by Squires and Quake (Squires and Quake, 2005). The two subclasses of droplet generators are flow-focusing droplet generators and T-shaped droplet generators. The syntax for declaring a droplet generator is given in Table 4.3. The key parameters are the width of the water flow channel, the width of the oil flow channel, and the radius of the internal ports. The width of the water channel should be at most 1/5 of that of the oil channel. The terminal numbering for a T-shaped droplet generator, `TDroplet` is

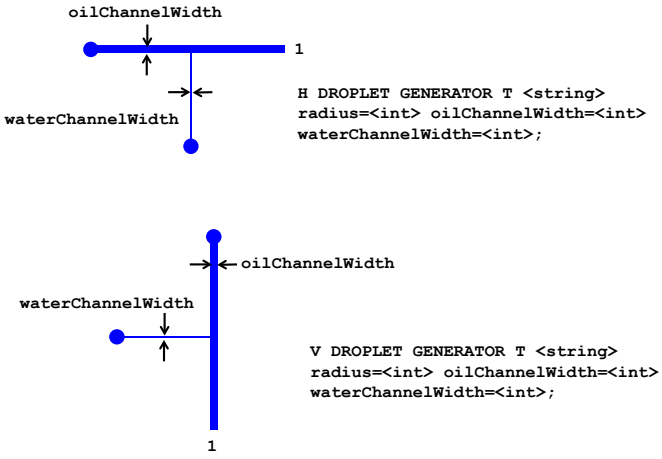


Figure 4·19: Diagram of parameters and terminal numbering for `TDroplet`. Droplets of the water from the narrow water channel are introduced to a stream of oil in the wider oil channel to form droplets. The width of the water channel should be at most 1/5 of that of the oil channel.

shown in Figure 4·19.

Additional parameters for the flow-focusing droplet generator, `FFDroplet`, include the angle of the connection between the water and oil flow channels and the length of the water flow channel as shown in Figure 4·20. This is one of the modules with an internal structure that would otherwise violate the architecture model as it allows channels to connect at angles other than right angles.

Rotary pump

The `RotaryPump` represents an active mixing structure as described in Urbanski et al. (Urbanski et al., 2006) where actuated valves around a ring mix the fluid inside the ring. The key parameters are the radius of the ring, the flow channel width, and the control channel width. The valves are sized according to the flow channel width. Parameters and terminal numbering are shown in Figure 4·21. The syntax for declaring a rotary pump is given in Table 4.3.

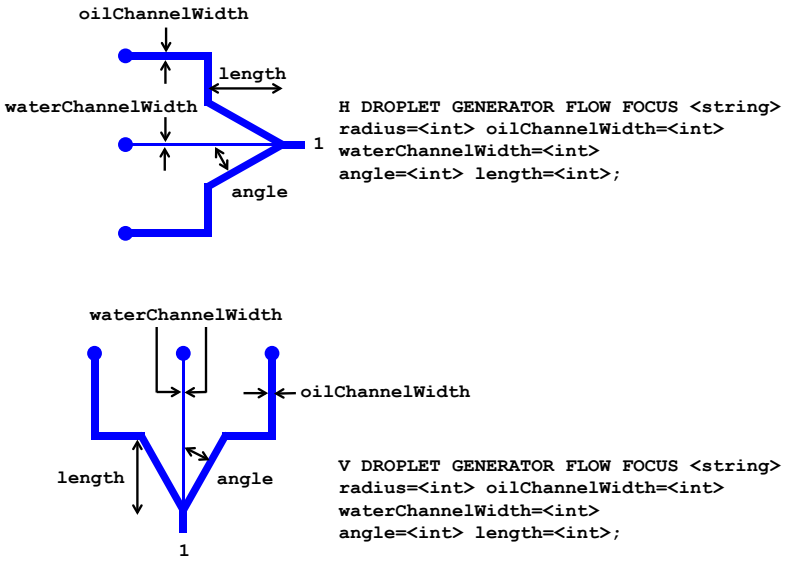


Figure 4·20: Diagram of parameters and terminal numbering for `FFDroplet`. Droplets of water are sandwiched between two flows of oil. Performance of the droplet generator is dependent on the angle at which the channels meet. The width of the water channel should be at most 1/5 of that of the oil channel.

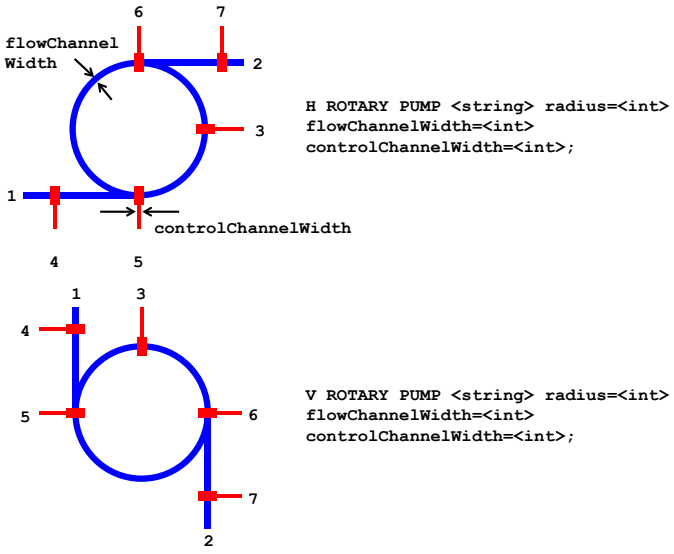


Figure 4·21: Diagram of parameters and terminal numbering for `RotaryPump`. The `RotaryPump` represents an active mixing structure as described in Urbanski et al. (Urbanski et al., 2006) where actuated valves around a ring mix the fluid inside the ring.

Table 4.4: Syntax for statements declaring 3D structures

Statement	Syntax
3D Valve	(‘V’ |‘H’) ‘3DVALVE’ `ID` ‘radius=’ `INT` ‘gap=’ `INT` ‘;’
Via	‘VIA’ `ID` (‘,’ `ID`)* ‘;’
Transposer	‘TRANSPOSER’ `ID` ‘valveRadius=’ `INT` ‘valveGap=’ `INT` ‘flowChannelWidth=’ `INT` ‘controlChannelWidth=’ `INT` ‘;’

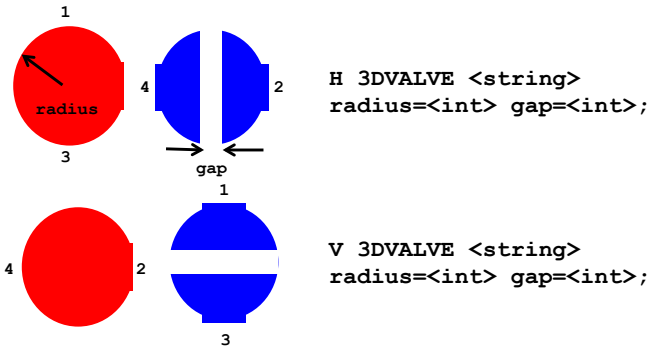


Figure 4·22: Diagram of parameters and terminal numbering for `Valve3D`. A layer of two-sided tape is used as the flexible membrane between the layers. The valve is closed normally, preventing flow. Negative pressure on the control layer causes the valve to open.

4.3.5 3D structures

The structures described in this section are for use with devices meant to be fabricated with a CNC milling platform or a method other than multilayer soft lithography. They may only be used in a device that has the `3D` flag. I represent these structures as modules as they span multiple layers. As such, they must be declared on the flow layer. Table 4.4 shows the syntax for declaring these structures in a device.

3D valve

The pneumatic valve structure for machined microfluidic devices, `Valve3D`, has a structure drastically different from that of valves for devices fabricated from multilayer soft lithograph. An overview of the mechanics of this valve is given in the following: (Duncan et al., 2015; Zhang et al., 2009; Becker and Gärtner, 2008).

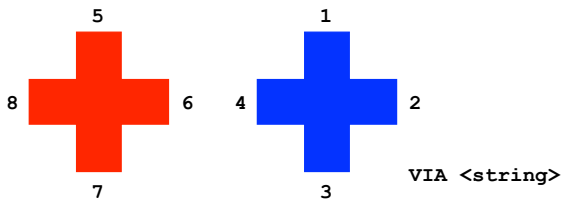


Figure 4·23: Diagram of parameters and terminal numbering for `Via`. The `Via` is a structure that allows channels on the flow layer to connect to channels on the control layer.

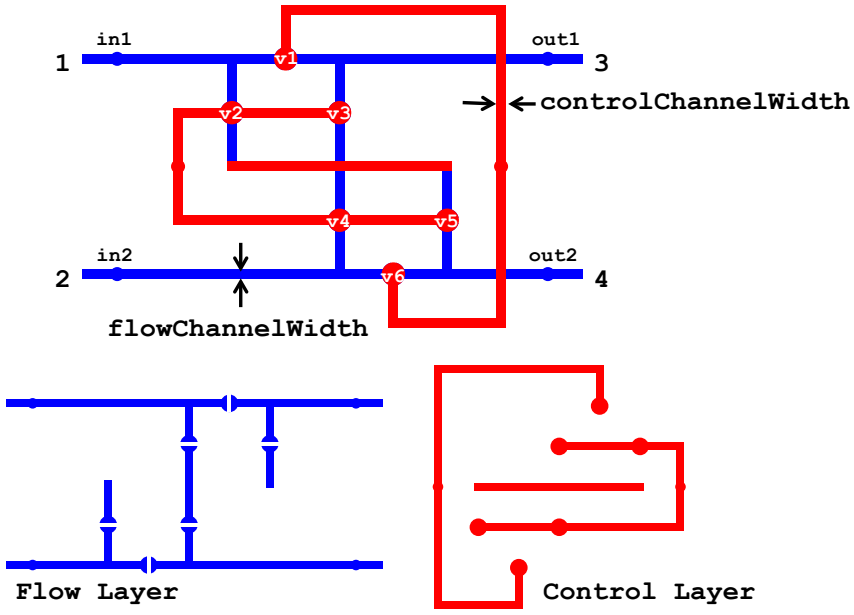
A layer of two-sided tape is used as the flexible membrane between the layers. The valve is closed normally, preventing flow. Negative pressure on the control layer causes the valve to open. The key parameters are the valve radius and the gap in the flow layer. The parameters and terminal numbering are shown in Figure 4·22. The syntax is given in Table 4.4.

Via

The `Via` is a structure that allows channels on the flow layer to connect to channels on the control layer. The terminal numbering is given in Figure 4·23, and the syntax is given in Table 4.4B. Multiple vias may be declared in the same statement.

Transposer

The `Transposer` is a structure designed to reconfigure flows between two inputs. When valves $v2$, $v3$, $v4$, $v5$ are closed, and valves $v1$ and $v6$ are open, the input at $in1$ connects to $out1$, and the input at $in2$ connects to $out2$. By opening $v2$, $v3$, $v4$, $v5$ and closing $v1$ and $v6$, $in1$ is redirected to $out2$ and $in2$ is redirected to $out1$. This structure moves the microfluidic design one step closer to the goal of a fully reconfigurable microfluidic device. The key parameters are the valve radius, the valve gap, the flow channel width, and the control channel width. Figure 4·24 shows the parameters and the terminal numbering. The syntax is given in Table 4.4C.




```
TRANSPOSER <string> valveRadius=<int>
valveGap=<int> flowChannelWidth=<int>
controlChannelWidth=<int>;
```

Figure 4·24: Diagram of parameters and terminal numbering for `Transposer`. The `Transposer` is a structure designed to reconfigure flows between two inputs. When valves $v2$, $v3$, $v4$, $v5$ are closed, and valves $v1$ and $v6$ are open, the input at $in1$ connects to $out1$, and the input at $in2$ connects to $out2$. By opening $v2$, $v3$, $v4$, $v5$ and closing $v1$ and $v6$, $in1$ is redirected to $out2$ and $in2$ is redirected to $out1$.

Chapter 5

Software workflow

In this chapter, I describe the software workflow as shown in Figure 5·1, including the input options, the algorithms for placement, routing, and design rule checking, and the output options. I will also discuss the current structure for adding biological function to microfluidic devices.

My workflow begins with two files, a microfluidic netlist file (as described in Chapter 4) and an initialization file with parameters for both microfluidic design rules. A layout is generated from the netlist after placement and routing. Design rules violations are noted by the design rule checker. The layout of the microfluidic device is output as either an `EPS` or `SVG` image file. An alternative workflow, similar to the ones described in Chapter 3, begins with the definition of a logic function in a hardware description language such as Verilog. The logic function is compiled by Cello (Vaidyanathan et al., 2015) into a set of biological logic in the form of a biological netlist. My workflow then generates a microfluidic device to house the biological logic and the control signals necessary to implement the function. The integration of the alternative test flow with Cello is still incomplete.

The program is written in Java 1.7 and compiled with Netbeans 8.0.1. Additional external libraries used by this program are listed in Appendix A.

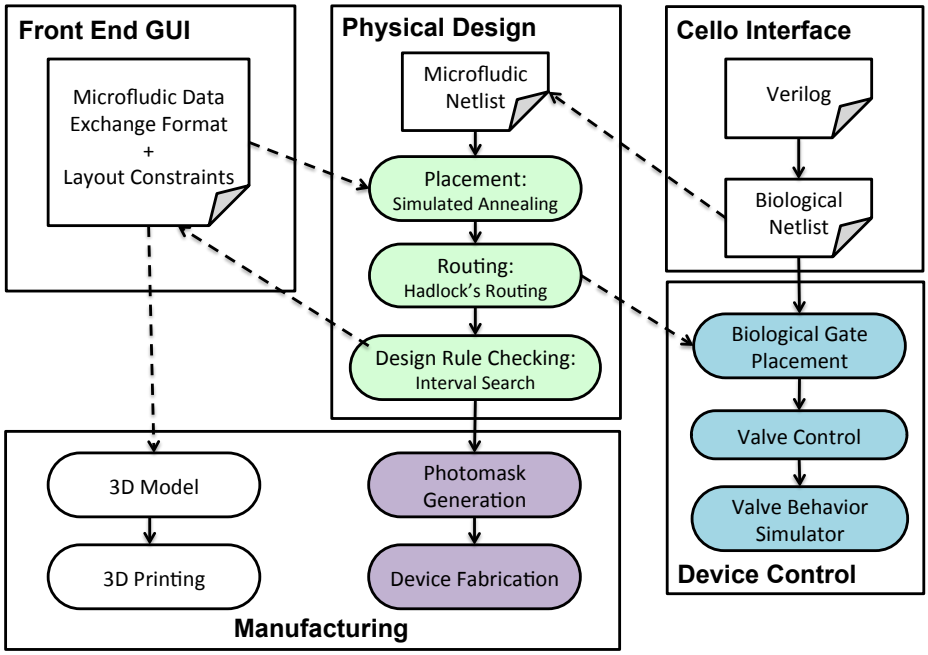


Figure 5·1: The workflow for Fluigi, from physical design to manufacturing and device controls. A device is represented as a data structure where each layer is a graph of channels and components such as ports, valves, and cell traps. A layout is generated from a set of physical constraints and a netlist describing the connections in the device using simulated annealing for placement (Betz and Rose, 1997) and Hadlock's algorithm (Hadlock, 1977) for routing. If given a set of biological devices that perform a function, Fluigi will generate a design of a device to house the biological devices and a set of valve control patterns to allow intercellular communications between them.

5.1 Initialization

I provide only a command line interface to run the software tool. The syntax for the command line is as follows:

```
java -jar fluigi filename [-i parameterFile] [-o output]
```

where `filename` is the name of the microfluidic netlist file including the extension of `.uf`, `parameterFile` is the name of the optional initiation file with the extension of `.ini`, and `output` is the output file format, either `eps` or `svg`. The default input parameters for `Fluigi` when no initialization file is used are shown in Tables 5.1 and 5.2. The default output file format is `eps`. The program will exit if either the input

Table 5.1: Default design rule parameters

Parameter Name	Soft Lithography (μm)	CNC (mm)	Description
minResolution	10	0.1	Minimum resolution of device features. Must be first parameter in file.
maxDeviceWidth	76800	500	Maximum width of finished device
maxDeviceLength	76800	500	Maximum length of finished device
routingSpacing	100	1	Space to reserve around components for channel routing
portSpacing	1000	1	Minimum space between adjacent ports and between ports and other device features
channelSpacing	40	0.1	Minimum spacing between two parallel channels. Channels may be on different layers.
valveSpacing	100	1	Minimum spacing between valves and any other device feature.
componentSpacing	100	1	Minimum spacing between all other components and any other device feature.

file or the parameters file, if it is used, does not exist. The initialization file also includes the output directory for designs. The default output directory is the current directory the program resides in.

Table 5.1 shows the default design rule parameters for devices manufactured with both traditional multilayer soft lithography and CNC milling. The parameters for soft lithography were derived from experimentation using the guidelines provided in Amin et al. (Amin et al., 2009). I define here the minimum feature resolution λ which I use as a scaling factor to determine the number of grid squares needed in grid routing. The default design rule parameters for CNC milled devices require further investigation for optimal results.

Table 5.2 shows the default settings for the placement and routing algorithms used in the workflow. Adjusting these settings will affect the likelihood of a design being successfully placed and routed. The placement and routing parameters do not differ based on manufacturing method as the design rule parameters do.

Table 5.2: Default place-and-route parameters

Parameter Name	Value	Description
iterations	20	Number of unsuccessful Place and Route iterations before program exits.
enableDRC	`true`	Enables design rule checking. Design rule failures return as failed place and route.
numMovesPerTempPerComponent	250	Number of times each component is moved at each temperature during simulated annealing.
overlapCost	10000	Factor to bias against component overlaps in simulated annealing.
areaCost	0	Factor to bias against devices with large area in simulated annealing.
channelLengthCost	1	Factor to bias against devices with long channels lengths in simulated annealing.
bendPenalty	2	Additional cost for bends in channels in grid routing.
spacingPenalty	5	Additional cost for violating spacing constraints in grid routing.
occupiedPenalty	10	Additional cost for using a grid square that is used in a previous layer.

5.2 Parsing the netlist file

The microfluidic netlist file is parsed by a parser and lexer generated from the netlist grammar file via `ANTLR3`. All measurements in the netlist file are given in μm, and are scaled by $1/\lambda$ during parsing. A `Device` object is generated at the end of parsing. Every `Component` has its location set to the point $(0, 0)$ unless the location of a `Component` has otherwise been set in the netlist file. A log file is then generated for the device which contains the detailed placement, routing, and design rule checking reporting.

5.3 Placement

I use a modified version of the simulated annealing algorithm as presented in (Betz and Rose, 1997) to generate a layout for the microfluidic device described by the netlist. I refer back to the definition of a `Device` as a graph $G(V, E)$ where each vertex $v \in V$ is a `Component`, and each edge $e \in E$ is a `Channel`. Graphs FV_f, E_f and CV_c, E_c where $V_f, V_c \subset V$ and $E_f, E_c \subset E$, representing the flow and control layers respectively, are subgraphs of G which are disconnected from each other. A `Module` m is a subgraph of G, and $m_1 \dots m_n \in M$ is the set of `Module`s in G. Let V_m be the set of all the `Component` in M, and E_m be the set of all `Channel` in M.

I add additional edges in G such that it is connected. More specifically, for every `Valve` $c \in V_c$, I find the source s and target t of the flow `Channel` $f \in E_f$, and add two edges in E to connect c to s and t. This ensures that valves specified on the control layer will be placed on the proper channel in the flow layer. Let $N_{components}$ be the number of `Component`, including `Module`, $v \in V \wedge \notin V_m$.

5.3.1 Simulated annealing

I calculate the initial temperature by making a random placement, where $\forall v \in V$, $v.x = random(0, 1) * maxDeviceWidth$ and $v.y = random(0, 1) * maxDeviceLength$, and perturbing the placement randomly a number of times, and set the initial temperature to 20 times the average cost of randomized placements. I use the cooling schedule described in (Betz and Rose, 1997) and stop the process when the temperature is a fraction of the average cost of a perturbation. The pseudocode for my simulated annealing process is shown in Figure 5·2.

I attempt to minimize the cost function

$$channelPenalty + overlapPenalty + areaPenalty$$

```
1: Calculate initial temperature: temp
2: Calculate initial cost: cost
3: rangeX = maxDeviceWidth
4: rangeY = maxDeviceLength
5: while temp > 0.005∗cost/N_components ∧ temp > 2 do
6:    for i = 0 to N_components ∗ numMovesPerTempPerComponent do
7:       select new Component c or Module m
8:       if random(-1,1) > 0 then
9:          c.x = c.x + random(-1, 1) * rangeX
10:      else
11:         c.y = c.y + random(-1, 1) * rangeX
12:      end if
13:      calculate new cost of placement
14:      if new cost < old cost then
15:         accept move
16:      end if
17:      if new cost > old cost then
18:         if random(0,1] < e^((old cost - new cost)/temp) then
19:            accept move
20:         else
21:            undo move
22:         end if
23:      end if
24:   end for
25:   rateAccept = % of moves accepted
26:   if rateAccept > 0.96 then
27:      temp = temp * 0.5
28:   end if
29:   if 0.8 < rateAccept ≤ 0.96 then
30:      temp = temp * 0.9
31:   end if
32:   if 0.15 < rateAccept ≤ 0.8 then
33:      temp = temp * 0.95
34:   end if
35:   if rateAccept ≤ 0.15 then
36:      temp = temp * 0.8
37:   end if
38:   rangeX = rangeX * (1.0 - 0.44 + rateAccept)
39:   rangeY = rangeY * (1.0 - 0.44 + rateAccept)
40: end while
```

Figure 5·2: Pseudocode for the simulated annealing algorithm. The temperature is updated according to the cooling schedule in (Betz and Rose, 1997).

Table 5.3: Channel length penalty calculations

Location of s_e	Location of t_e	Penalty if
top	bottom	$s_e.y > t_e.y$
bottom	top	$s_e.y < t_e.y$
right	left	$s_e.x > t_e.x$
left	right	$s_e.x < t_e.x$

$$channelPenalty = channelLength * channelCost + numPenalty * overlapCost$$

$$areaPenalty = area * areaCost$$

$$overlapPenalty = overlapArea * overlapCost$$

where *channelLength* is the sum of the half-perimeter length of all channels $e \in E \wedge \notin E_m$, *area* is the total area of the device, and *overlapArea* is the total overlap area of all components $v \in V \wedge \notin V_m$. I currently ignore the area of the device ($areaCost = 0$) and bias heavily against overlaps ($overlapCost = 10000$) such that the length of all channels $e \in E \wedge \notin M_e$ are minimized. In the default scenarios, I set $channelCost = 2$. For the purposes of calculating the cost function, I treat all nets as a single channel between the source and the first target.

I calculate *channelLength* with the following:

For channel $e \in E \wedge \notin E_m$, let s_e be the source terminal and t_e be the target terminal of channel e. Let $p_s = x_s, y_s$ be the coordinates of the source terminal and $p_t = x_t, y_t$ be the coordinates of the target terminal.

$$e.length = abs(x_s - x_t) + abs(y_x - y_t)$$

I add a penalty to each channel where the relative positions of s_e and t_e will likely cause routing problems. A terminal can be located on the top, bottom, left, or right of a component. The cases where the terminals are positioned such that routing will likely have to go around one or more components are shown in Table 5.3.

For calculating both area and overlap, I included the required spacing around each

component (*spacing*) and spacing allotted for routing (*routingSpacing*) as described in Table 5.1.

For `Component` $v_1 \dots v_n \in V \wedge \notin V_m, n = N_{components}$:

$$v_i.left = v_i.x - v_i.spacing - routingSpacing$$

$$v_i.right = v_i.x + v_i.width + v_i.spacing + routingSpacing$$

$$v_i.top = v_i.y - v_i.spacing - routingSpacing$$

$$v_i.bottom = v_i.y + v_i.length + v_i.spacing + routingSpacing$$

The area of the device layout is calculated by the following:

$$area = (max(v_i.right) - min(v_i.left)) * (max(v_i.bottom) - min(v_i.top))$$

I consider two components v_a and v_b to overlap if:

$$v_a.left \leq v_b.right \wedge$$

$$v_a.right \geq v_b.left \wedge$$

$$v_a.bottom \leq v_b.top \wedge$$

$$v_a.top \geq v_b.bottom$$

The overlap area between v_a and v_b, should they overlap, would be:

$$overlapArea = overlapX * overlapY$$

$$overlapX = (min(v_a.right, v_b.right) - max(v_a.left, v_b.left))$$

$$overlapY = (min(v_a.bottom, v_b.bottom) - max(v_a.top, v_b.top))$$

I then find in the layout all instances of overlaps and calculate the total overlap

area.

5.3.2 Pre-routing layout cleanup

I adjust the postions of the components after simulated annealing to align terminals of components to ensure minimal bends in channels during routing. To straighten channels, I adjust the source and target of each channel such that the channel is straight if possible. To minimize bends in channels between two modules, I center modules with each other to align the terminals. I find all channels between the two modules and calculate the average (x, y) coordinates of the source terminals on one module and all of the target terminals on the other module. If all the connections are horizontal, I move the modules such that that $source.y_{center} = target.y_{center}$. If all the connections are vertical, I move the modules such that $source.x_{center} = target.x_{center}$. Finally I move all flow and control ports not in a bank to the edges of the device to allow better access during manufacturing and test. Ports are moved to the nearest edge of the device based on which direction the port is connected. For example, ports with a connection on the top are moved to the bottom edge. For all the above cases, no moves are made if they cause any overlaps.

5.4 Routing

After placement, I route all channels and nets with Hadlock's algorithm (Hadlock, 1977), a grid based maze router. I based my implementation on the source code for `Grid.java` and `GridPoint.java` found at `http://workbench.lafayette.edu/~nestorj/cadapplets/HadlockRouter/HadlockRouter.html` (Nestor, 2007). The pseudocode is shown in Figure 5·3. I set the length and width of each grid square to be λ, and generate a routing grid the size of the layout after placement with an additional border of extra space for routing, the size of which is given by the parameter $routingBorder$. At initialization, each grid square is assigned one of four values for

its base cost:

- Empty and available for routing: $cost = 1$
- Within minimum spacing of a channel or component: $cost = spacingCost$;
- Occupied by a channel or component on a different layer: $cost = occupiedCost$;
- Blocked by a channel or component on the current layer being routed: $cost = -1$

When routing, I treat all grid squares with a cost of -1 as obstacles to route around. I also introduce a bend penalty to the routing cost calculation to minimize the number of bends in the path for a channel.

The flow layer is routed first, followed by the control layer. On each layer, I first route all nets, followed by all channels not in nets. Currently, nets may only be declared on the control layer.

5.4.1 Channel routing

For each channel to be routed, I find the grid location of the source and target terminals. I then identify the grid square closest to the target terminal that is not within the spacing requirement of the component, and designate that grid square as an intermediate target. This lessens the chance of violating the spacing requirement of the target component. I use Hadlock's algorithm with the added bend penalty and the aforementioned grid square costs to find a lowest cost path between the source and the intermediate target, and between the intermediate target and the final target. As each new grid square is encountered, it is updated with the cost of the path to reach itself and which direction (up, down, left, or right) the expansion to reach it took place. I use this information to backtrack a path from the target terminal to the source.

```
1: Input: grid of M x N with initial obstructions marked, list of channels to be routed in
   each layer
2: Output: grid with all channels routed, if possible
3: for layer in device do
4:    for channel in list of channels to be routed do
5:       current = channel.sourceTerm
6:       target = channel.targetTerm
7:       minPriorityQueue = ∅
8:       currentDetourCost = 0, currentPathCost = 0
9:       minPriorityQueue.add(current)
10:      while current ≠ target and minPriorityQueue ≠ ∅ do
11:         current = minPriorityQueue.pop
12:         for neighbors of current do
13:            detourCost = currentDetourCost + ManhattanDistance(neighbor, target)
14:            if neighbor causes bend in path then
15:               pathCost = currentPathCost + neighbor.baseCost + bendCost
16:            else
17:               pathCost = currentPathCost + neighbor.baseCost
18:            end if
19:            if neighbor is not expanded and neighbor.cost ≤ current.cost then
20:               minPriorityQueue.add(neighbor)
21:            end if
22:         end for
23:      end while
24:      if target is found then
25:         Backtrack to source via lowest cost path
26:      end if
27:      Clear all cell costs
28:   end for
29: end for
```

Figure 5·3: Pseudocode for Hadlock's routing algorithm with added bend penalty.

When routing to or from a `Node` I first set the grid locations of any previously routed channels connected to the node to $cost = 1$. I use nodes to substitute for nets on the flow layer, and this allows me to mimic the behavior of multiterminal net routing.

Routing fails if the target terminal cannot be reached. In that case, the program will clear the routing grid and the layout used to generate the grid, and begin with a new placement. As Hadlock's algorithm degenerates to exhaustive search in the

worst case scenario, a path between the source terminal and target terminal will be found if it exists.

After a channel is successfully routed, I go through the path and reduce it to only the points where a direction change occurs. I then modify the routing grid to include the location of the channel. All grid squares in the path and within $channelWidth/2$ of the path are set to $cost = -1$. All grid squares between $channelWidth/2$ and $channelWidth/2 + channelSpacing$ of the path are set to $cost = spacingCost$.

Once all channels on a layer have been routed, I set the locations of those channels on the routing grid to $cost = occupiedCost$ before routing the next layer. As my defintion of channel previously did not include changes of direction or bends within the path of a channel, I decompose each channel with bends into a set of channels and nodes, with the bends occurring at the nodes. I remove the orignal channel from the device and add the new channels and nodes.

5.4.2 Net routing

For each net to be routed, I proceed as I do with channel routing between the source terminal and the first terminal in the list of target terminals. For all other terminals, I set the source for the expansion not as the source terminal but as the previously routed path. All else proceeds as in channel routing, including decomposing the path into individual channels once routing is successful.

5.5 Design rule checking

I perform two types of design rule checking after a device has been successfully routed. If the device does not have the `3D` flag, I check for all overlaps between channels on the flow layer and channels on the control layer. For every control channel that crosses a flow channel, I narrow the width of the control channel to $20\mu m$ where it crosses the flow channel. This is to prevent the control channel from forming a seal across

the flow channel when it is not supposed to. This operation is not needed for device with the `3D` flag due to differences in valve construction between devices fabricated with multilayer soft lithography and with CNC milling.

For all devices, I verify that all design rule parameters as specified in the initiation file are met. Devices that violate design rule parameters are treated as having failed routing and go through further iterations of placement and routing until all design rule checks are successful.

5.5.1 Channel intersection checking

I use a 1-D range check algorithm (Sedgewick and Wayne, 2015; Berg et al., 2008) to search for intersections between flow channels and control channels. The pseudocode for this is shown in Figure 5·4. I check for intersections between horizontal flow channels and vertical control channels and between vertical flow channels and horizontal control channels. If an intersection is found, I add to the control channel the location of the crossing. This is taken into consideration when the control channel is being rendered, and the control channel will be narrowed to $20\mu m$ at the intersection for the width of the flow channel plus extra space on either side equal to *channelSpacing*.

I first sort all channels by the x coordinate of their leftmost end point. I define an `Event` to have a `time` and a `Channel`, where `time` is the x coordinate of the end point of a channel. For each horizontal channel, I add two `Event`s to the `EventQueue` for the left and right end points of the `Channel`. The `EventQueue` is implemented with a priority queue that sorts by `time` in descending order from smallest to largest. For each vertical channel, I add one `Event` to the `EventQueue`.

At the same time, I create a Binary Search Tree (BST) `RangeSearch` (`http://algs4.cs.princeton.edu/93intersection/RangeSearch.java.html`). The BST is sorted based on the y-coordinate `EventQueue` is polled until it is empty. For each `Event` that maps to the left end point of a horizontal channel, I add the channel to

```
 1: eventQueue = ∅
 2: for channel in device do
 3:    if horizontal channel then
 4:       eventQueue.add(channel.minX, channel)
 5:       eventQueue.add(channel.maxX, channel)
 6:    else
 7:       eventQueue.add(channel.x, channel)
 8:    end if
 9: end for
10: setBST = ∅
11: while eventQueue ≠ ∅ do
12:    event = eventQueue.getMin
13:    sweep = event.x
14:    segment = event.channel
15:    if sweep = segment.minX then
16:       setBST.add(segment)
17:    end if
18:    if sweep = segment.maxX then
19:       setBST.remove(segment)
20:    end if
21:    if vertical segment then
22:       line 1 from (-∞, segment.minY) to (∞, segment.minY)
23:       line 2 from (-∞, segment.maxY) to (∞, segment.maxY)
24:       for channel in setBST.range(line 1, line 2) do
25:          intersection with segment
26:       end for
27:    end if
28: end while
```

Figure 5·4: Pseudocode for 2D intersection search algorithm.

the BST. For each `Event` that maps to the right end point for a horizontal channel, I remove the channel from the BST. For each `Event` that maps to a vertical channel, I search the BST for all horizontal channels with a y-coordinate that falls between the end points of the vertical channel. I then mark all intersections and add those intersections to the appropriate control channel.

```
1: eventQueue = ∅
2: for element in device do
3:    eventQueue.add(element.minX, channel)
4:    eventQueue.add(element.maxX, channel)
5: end for
6: intervalBST = ∅
7: while eventQueue ≠ ∅ do
8:    event = eventQueue.getMin
9:    sweep = event.x
10:   segment = event.element
11:   if sweep = element.minX then
12:      intervalBST.findAllIntersections(interval(element.minY, element.maxY))
13:      for intersection found do
14:         verify spacing
15:      end for
16:      intervalBST.add(interval(element.minY, element.maxY))
17:   end if
18:   if sweep = element.maxX then
19:      intervalBST.remove(interval(element.minY, element.maxY))
20:   end if
21: end while
```

Figure 5·5: Pseudocode for spacing verification interval search algorithm.

5.5.2 Space constraint checking

I ensure all design rules for spacing contraints around features (components and channels) in a device are enforced. A design rule violation manifests as an intersection between two features. To find all intersections between features, I compare the location of each feature, including the minimum and maximum x and y coordinates, with every other feature to find any violations in spacing. For a device $G(E, V)$ with E channels and V components, an exhaustive search would take $O((E+V)^2)$. I instead reduce the problem of finding 2-D intersections to a 1-D interval search (Sedgewick and Wayne, 2015; Berg et al., 2008). The pseudocode for this is shown in Figure 5·5.

To accomplish this, I create an interface `Element` that both `Component` and `Channel` implement. This interface accesses the minimum and maximum x and y

coordinates of a `Component` or a `Channel` defined as:

$$minX = element.x - element.spacing$$

$$maxX = element.x + element.width + element.spacing$$

$$minY = element.y - element.spacing$$

$$maxY = element.y + element.length + element.spacing$$

I sort all elements by first by $minX$, then by $maxX$ and by $minY$. For each element, I create an `Interval` from $minY$ to $maxY$. Here, I define an `Event` to have an `Element`, an `Interval`, and a time. For each element f, I create two `Event`, one at time $f.minX$ and one at $f.maxX$. I add each event to the `EventQueue`, a priority queue sorted by time in descending order.

To find potential intersections, I poll the `EventQueue`. If the event has a time equal to $f.minX$, I add the interval to the interval search tree. The interval search tree is a modified BST where each node stores an interval as the key. The intervals in the tree are sorted by $minY$, and each node also stores the maximum $minY$ value of its subtrees. I based my implementation on the interval search tree code at `http://algs4.cs.princeton.edu/93intersection/IntervalST.java.html`. When an interval is added to the interval tree, I search the tree for any overlaps with intervals already stored in the tree, and return the list of overlaps.

For each overlap between two elements f_a and f_b, I check to see if the overlap results in a design rule violation. The overlap is not a design rule violation if:

- The overlap between f_a and f_b is less than $max(e_a.spacing, e_b.spacing)$.
- f_a and f_b are a component and a channel connected to that component
- f_a and f_b are a control channel and flow channel that overlap, and that overlap

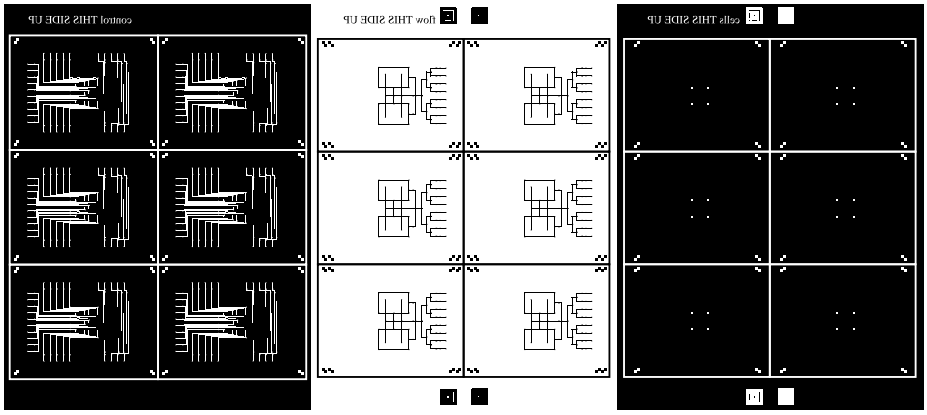


Figure 5·6: The final rendered photomasks for a design made with Fluigi with three layers, the flow layer, the 'cell' layer for feature heights on the flow layer, and the control layer. Large alignment marks are on the top and bottom masks for flow and cell layers, and small alignment marks are on all layers in the corners of each device.

has been accounted for during channel intersection checking.

- f_a and f_b are a valve and a flow channel that valve is associated with

I output in the log file a list of all design rule violations in the device, including the amount of overlap in each violation. Devices with design rule violations are treated as having failed routing, and go through further place-and-route iterations unless the `enforceDRC` parameter is set to `false`.

5.6 Photomask generation

Once design rule checking is complete and returns with no violations, I render the design of the device in either an `EPS` or `SVG` file. I assume that designs rendered as `EPS` are used as photomasks for multilayer soft lithography, and designs rendered as `SVG` are used for CNC milling.

5.6.1 Masks for photolithography

When rendering the design into `EPS`, I render the flow layer, which may have two photomasks for different feature heights, and the control layer in the same image file. I add a buffer zone of 1mm around each device, and then calculate the maximum number of devices that will fit into a 4 inch wafer and tile the designs. Guidelines for cutting are drawn between each tile. An example of a set of photomasks with three masks for the flow layer, additional feature heights in the flow layer, and the control layer is shown in Figure 5·6. In addition, I add alignment marks to the corners of each tile, as shown in Figure 5·7, such that the flow layer and control layer may be aligned during fabrication. Larger alignment marks, a closeup of which is shown in Figure 5·8, will be added to the top and bottom of the flow layer photomasks for mask aligning when making the flow layer mold.

The control layer uses a negative photoresist, and the mask is drawn with white features on a black background. The flow layer uses a positive photoresist, and the mask is drawn with black features on a white background. Additional flow layer masks for different feature heights use negative photoresists and are drawn like the control layer. All masks are drawn as a horizontal reflection. The text 'THIS SIDE UP' is added to the top of each mask to ensure masks are loaded into the mask aligner appropriately. All masks are drawn in the same file.

5.6.2 Designs for CNC milling

Each layer is drawn in a separate file. The control layer is reflected horizontally, but the flow layer is not. No alignment marks are needed.

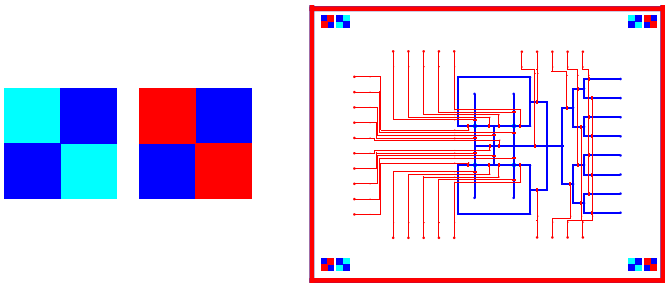


Figure 5·7: A closeup of the small device alignment marks. Dark blue marks are on the flow layer. Red marks are on the control layer. Cyan marks are on the flow layer for different feature heights. Each alignment square is 0.5mm by 0.5mm. Each device has 4 sets of small alignment marks.

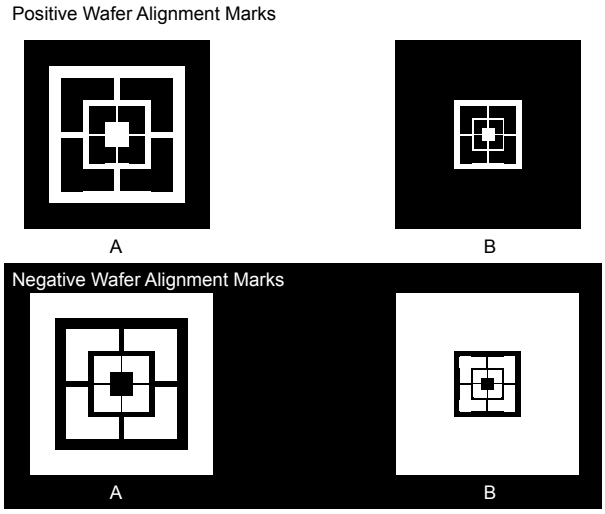


Figure 5·8: A closeup of the large alignment marks. The `A` marks have 4 rectangles of 3mm by 0.4mm, 8 rectangles of 0.85mm by 0.35mm, and 8 rectangles of 0.43mm by 0.25mm. The `B` marks have 4 rectangles of 3mm by 0.9mm, 8 rectangles of 0.43mm by 0.18mm, and 8 rectangles of 0.21mm by 0.12mm

5.7 Adding biology

In addition to the design flow from microfluidic netlist through photomask generation, I incorporate an additional design flow from the description of a Boolean logic function in a hardware description language to a device that can house the biological network that performs that function. This design flow is still incomplete as it does not interface to the program Cello (Vaidyanathan et al., 2015) that converts the hardware design

language to a biological network. I assume from this point forward that the program will receive as input a biological network that can be described as a directed acyclical graph $B(V, E)$ where vertices are cells that embody a particular logic function or input and output signals, and edges are extracellular signals used by the cells to communicate or other small signalling molecules. In addition, the work presented in this section is only applicable to devices made from multilayer soft lithography.

I modify my existing model for microfluidic components to include additional methods for components that can house cells or signals. I create an interface `Fluid` to represent cells, media, signalling modules, or other liquids, which is implemented by two classes, `Gate` and `Signal`. The class `Gate` represents one colony of cells that perform a specific logic function. A `Gate` has a name, a two-input logic function represented as a string (`AND`, `OR`, `NOR`, `NAND`, `NOT`), and a cell type, as it may be possible to have multiple colonies from one type of cell or multiple cell types implementing the same logic function. At the moment, I represent cell types, media, waste, and small signalling molecules with `Signal`. I also create an interface `Chamber`, implemented by `Port`, `CellTrapL`, and `CellTrapS`, which associates a `Fluid` with a microfluidic component designed to house it.

If the biological network contains four gates or fewer, I use a device centered around the logic array (Chapter 4.3.4) to house the network. Otherwise, I refer to the design paradigm introduced in Chapter 3.3.2 to generate a microfluidic device to house the biological function. In both cases, I assign an input port to every different cell type, and input signal, and a cell trap to every `Gate`. I also assign media and waste ports to each `Gate`. Let g be a `Gate` with a function f, a cell type of *cell*, input signals a and b, output signal c. I will refer to the cell trap housing g as ct_{gate}. Port p_{cell} contains the cell type, and ports p_a and p_b contain the input signals. Port p_{media} and p_{waste} provide media and remove waste from ct_{gate} respectively.

Table 5.4: Control pattern operations

Operation	Path
Load cells	p_{cell} to ct_{gate} to p_{waste}
Load input	$input$ to ct_{gate} to p_{waste}
Grow cells	p_{media} to ct_{gate} to p_{waste}
Flush output	p_{media} to ct_{gate} to $output$

5.7.1 Generating controls

I determine the number of ports on the control layer, and the number of valves connected to each port. I assume that all valves are by default closed (pressure being applied), blocking flow. For every `Gate` in the function to be implemented, I find the following paths in the device :

- p_{cell} to ct_{gate}
- p_{media} to ct_{gate}
- ct_{gate} to p_{waste}

In conjunction with graph $B(V, E)$, I also find paths from chambers housing inputs of g to ct_{gate} and from ct_{gate} to chambers housing outputs of g. Pathfinding is accomplished with Djikstra's shortest path, and returns the list of channels forming the shortest path. For the logic array, I have modified the weights of some channels to prevent paths from passing through additional cell traps.

I define the following operations and paths for each gate:

For each channel in a path, I determine if a valve exists to control the channel. I open all valves in a path when each operation is performed. The timing of the opening and closing of the valves depends on the specifics of each cell type.

To initialize the experiment, I open all the valves in the device and fill the device with a buffer solution. I then close all the valves. Next, I load cells in each cell trap, and grow them. For each pipelined stage of the logic function, I apply inputs to the

gates in that stage, grow the cells until the logic is processed, and flush the output to the next stage in the computation.

I generate Java files to interface to the existing control software BioStream, which was developed to design GUIs and control valves for multilayer devices (Thies et al., 2008; Amin et al., 2009; Urbanski et al., 2006) (freely available at (Thies et al., 2009)). These files map each control line to one of the 48 physical ports on the control apparatus and define the mechanisms for opening and closing a port and for closing or opening all the ports on the chip.

Chapter 6

Results

In this chapter, I show a selection of designs generated by my software and discuss the runtime and efficiency of the workflow I presented in Chapter 5. I also show two examples of the end-to-end workflow, one resulting in a CNC machined device, and one resulting in a multilayer soft lithography device.

6.1 Example designs

I use the netlist format described in Chapter 4 and the software workflow to generate a selection of microfluidic device designs. Each feature described in the netlist is shown in at least one of these devices. The full netlists for these devices are shown in Appendix B. In figures of the device designs, all the layers are overlaid, with blue representing the channels and ports on the flow layer, cyan representing features of a different height on the flow layer, and red representing the features on the control layer.

6.1.1 Designs for multilayer soft lithography

Table 6.1 shows the device sizes and runtimes of the 11 devices, from least complex to most complex, for use with traditional fabrication methods. The devices themselves are shown in Figures 6·1 - 6·4. For these devices, I used the default initialization and place-and-route parameters as shown in Tables 5.1 and 5.2. In particular, the minimum resolution λ was set to 10 μm, and the number of moves per component

Table 6.1: Examples of devices for multilayer soft lithography

Name	**Number of Components**	**Number of Channels**	**Nets**	**Total Elements**	**Runtime (hr:min:sec)**	**Device Width (cm)**	**Device Length (cm)**
A (Figure 6·1)	4	3	0	7	0:00:05	0.99	1.2
B (Figure 6·1)	9	6	0	15	0:00:07	1.27	1.13
C (Figure 6·1)	9	8	0	17	0:00:10	1.77	1.91
D (Figure 6·1)	9	17	0	26	0:00:37	4.24	1.31
E (Figure 6·2)	16	16	0	32	0:00:25	2.79	1.52
F (Figure 6·2)	10	23	0	33	0:00:22	2.69	1.06
G (Figure 6·2)	13	28	0	41	0:00:56	3.64	1.63
H (Figure 6·3)	9	41	0	50	0:01:34	3.58	2.18
I (Figure 6·3)	11	55	0	66	0:02:41	3.36	2.93
J (Figure 6·4)	61	67	2	130	0:02:47	2.3	2.76
K (Figure 6·4)	183	194	8	385	0:12:49	2.02	3.5

per temperature was set at 500.

I characterize the complexity of these devices based on the total number of features, including the number of components, the number of channels, and the number of nets. Modules (as described in Chapter 4.3.4), are treated as a single component. The number of elements ranges from 7 to 385, and the runtimes range from 5 seconds to 12:49. The runtime appears to be $O(n^2)$ where n is the total number of features on a device. Runtime is affected by the physical size of devices as well as the number of elements, as channels on larger devices take longer to route.

Devices `A`, `B`, `C` and `D` from Table 6.1 are shown in Figure 6·1. These are the least complex devices I made using my tool. Device `A` contains a flow focusing droplet generator that connects to a section of channel with a reduced channel width. However, I was not able to specify the exact length of channel with my current netlist format. Device `B` has a cell trap with two inputs, each controlled with a valve, and device `C`

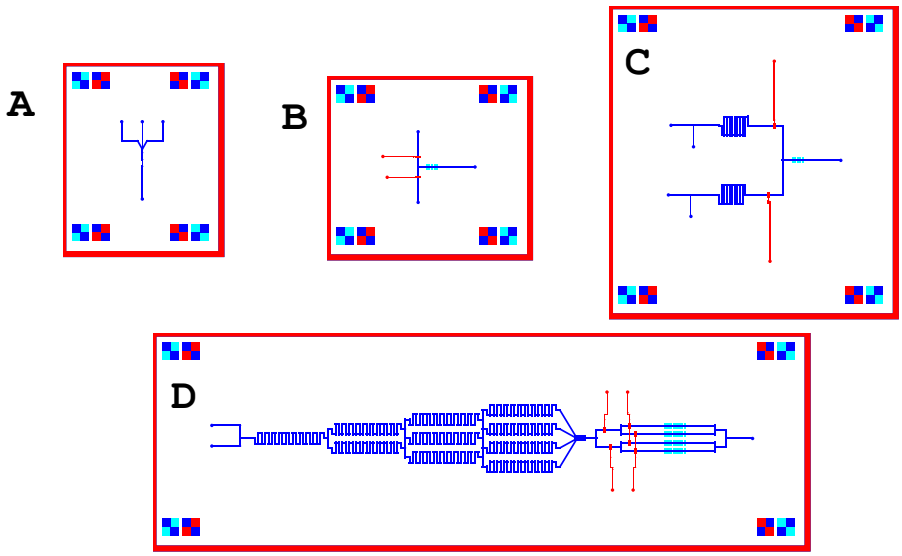


Figure 6·1: Device `A` is an example of a flow focusing device. Device `B` is a simple device that allows switching of two inputs into a cell trap. Device `C` has a cell trap that switches between two t-shaped droplet generators. Device `D` has a gradient generator whose output can be directed to any of four cell traps. The statistics and runtimes for these devices are shown in Table 6.1.

has a cell trap that can switch between two droplet generators as inputs. Device `D` has a gradient generator connected to a bank of 4 cell traps. It has a longer runtime than device `E` (shown in Figure 6·2) despite being less complex because the range of component sizes (from $100\mu m$ by 100 μm to $2cm$ by 0.6 cm, spanning 2 orders of magnitude) in device `D` is suboptimal for the collision detection hash table used during placement.

Device `E` has one central square cell trap with a choice of 4 inputs, 2 of which come from mixers. Device `F` is based on the microfluidic device used to test oscillations in gene expression of *E. coli* (Prindle et al., 2012) and has a large bank of closely placed cell traps. Control of fluid flow is achieved through pressure differentials instead of valving. Device `G` has a rotary pump with up to 4 different inputs which can be connected to any of four cell traps.

Device `H` is based on the logic array module and has 8 inputs, each of which can

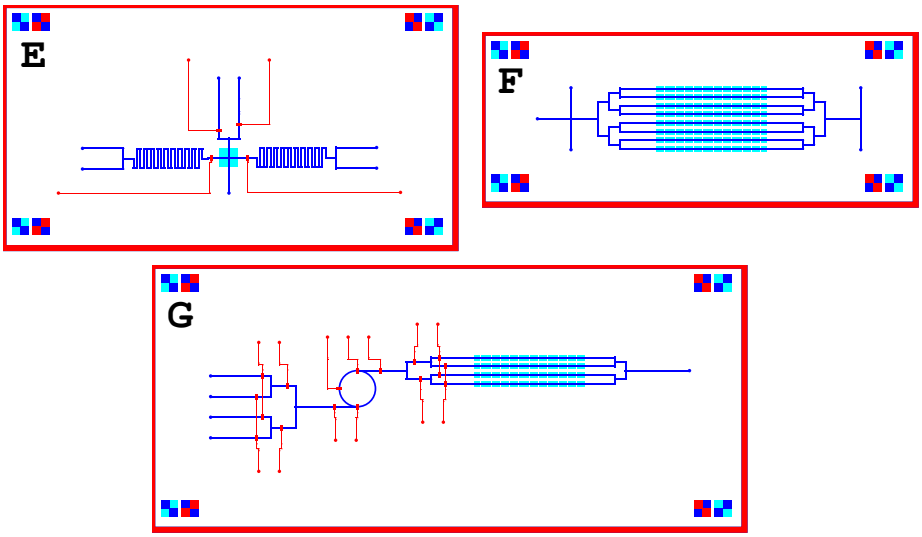


Figure 6·2: Device `E` has one large cell trap with a choice of four different inputs. Device `F` is a modified version of the cell grids from the Hasty lab (Prindle et al., 2012). Device `G` has a rotary with four potential inputs whose output can be directed to any of four cell traps. The statistics and runtimes for these devices are shown in Table 6.1.

be directed to any of the chambers, and device `J` has a rotary pump that can switch between any of 16 inputs and 16 outputs. Both these devices depend heavily on the use of modules, which is why they have comparatively fewer components than channels. I use device `H` as a potential microfluidic device for replicating the `NOR` gate experiment conducted by the Voigt lab (Tamsir et al., 2011).

Devices `J` and `K` both have large numbers of cell traps and inputs to allow for multiple experiments on the same device. In device `J`, each cell trap can be exposed to three different inputs to produce various outputs while in device `K`, each column of cell traps can be isolated from the rest with valving. These are the two most complicated devices presented in this section, with 130 and 385 components respectively.

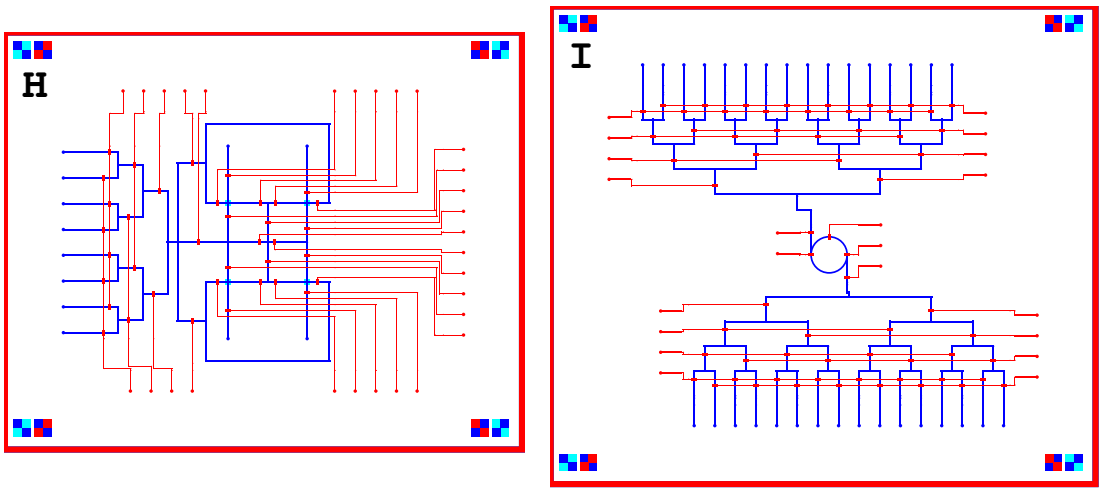


Figure 6·3: Device `H` is based on the logic array module with 8 inputs. Device `I` is a rotary pump with 16 inputs and 16 outputs. The statistics and runtimes for these devices are shown in Table 6.1.

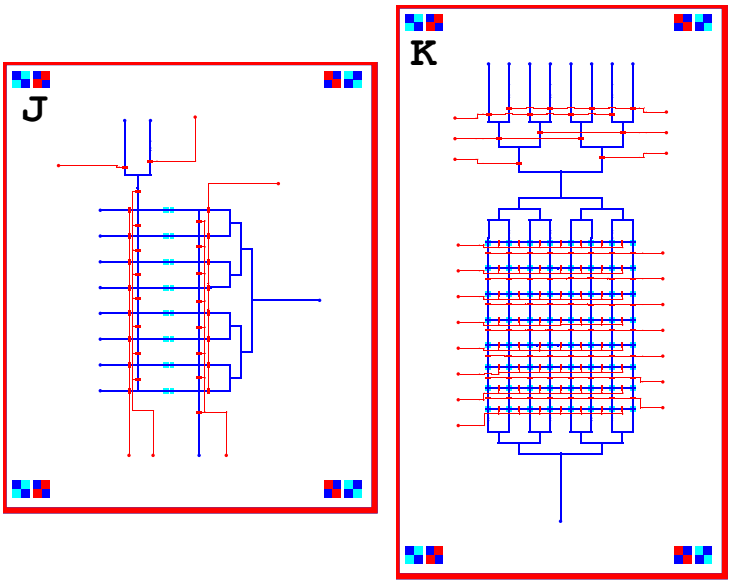


Figure 6·4: Device `J` has 8 cell traps, with the selection of 3 different inputs for each cell trap. Inputs may be applied simultaneously or individually. Device `K` has a grid of 64 cell traps and 8 possible inputs. The statistics and runtimes for these devices are shown in Table 6.1.

Table 6.2: Examples of devices for CNC machining

Name	**Number of Components**	**Number of Channels**	**Total Elements**	**Runtime (hr:min:sec)**	**Device Width (cm)**	**Device Length (cm)**
transposer-web	5	6	11	0:00:20	14.82	7.8
txtl	7	6	13	0:00:06	2.97	4.24
transposer-mixer	7	10	17	0:00:12	10.31	4.42
mixer-3d	9	11	20	0:00:14	5.12	14.61

6.1.2 Designs for CNC machining

I also generated 4 designs for use with CNC milling instead of photolithography. As with the previous devices, I used the default parameters for initialization and place-and-route. For these devices, I set the minimum resolution to 100 μm instead of 10 as the CNC mill cannot provide the same level of feature resolution as photolithography. The statistics and runtimes for these devices are shown in Table 6.2, and the device designs are shown in Figure 6·5. As the resolution is larger, I can generate much bigger devices without sacrificing runtime.

The first of the devices (`txtl`) is a mixer connected two ports, each controlled by a valve. This was a theoretical device used to test the ability to mix cell free transcription/translation solution. Devices `transposer_web` and `transposer_mixer` show some of the potential configuration for connecting transposers with other components. Finally, the device `mixer_3d` consists of a gradient generator connected to three inputs, where each input is controlled by a valve. This device was used as a test case for the CNC mill fabrication workflow.

6.2 Algorithm runtimes

To test the efficiency and runtimes of the software, I wrote a Python script to generate netlists of square cell traps and valves connected in a grid pattern. Examples of these grid devices (4x4, 8x8, and 15x15 grids of cell traps) are shown in Figure 6·6. I

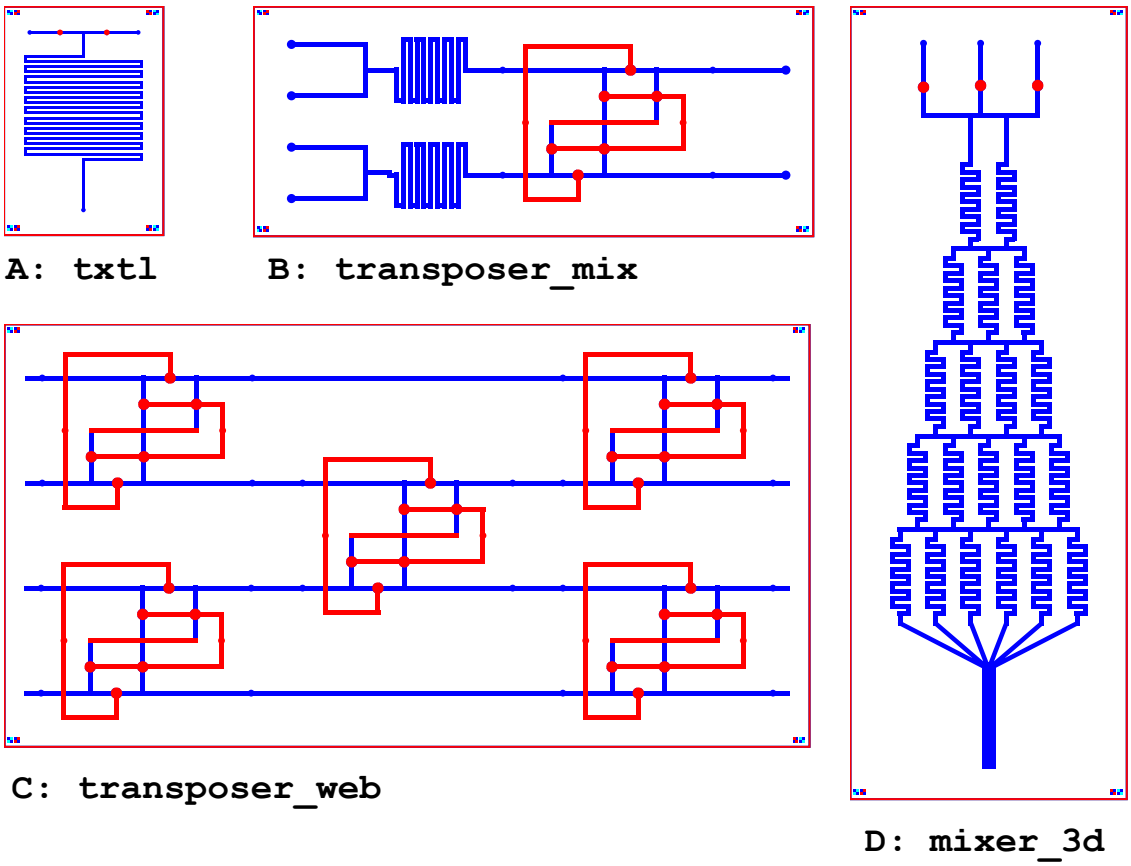


Figure 6·5: Device `A` shows a device used to mix cell-free transcription/translation solution. Device `B` shows a transposer connected to two mixers. Device `C` shows a connected web of 5 transposers that would allow switching between 4 different input flows. Device `D` shows a gradient generator with three selectable inputs. The statistics and runtimes for these devices is shown in Table 6.2.

generated netlists for 13 test devices, from a 4x4 grid to a 16x16 grid, and ran my software on those netlists. The results and runtimes are shown in Table 6.3. I used the default initialization and place-and-route paratemters with one exception. The number of moves per component per temperature step was set to 250 instead of 500.

The number of components ranges from 96 in the 4x4 grid to 1510 in the 16x16 grid. The runtimes range from forty-nine seconds to two hours twenty-five minutes and twelve seconds. Closer examination of runtimes show it to be between $O(n^2)$ and $O(n^3)$ where n is the number of features in the device. As the number of components

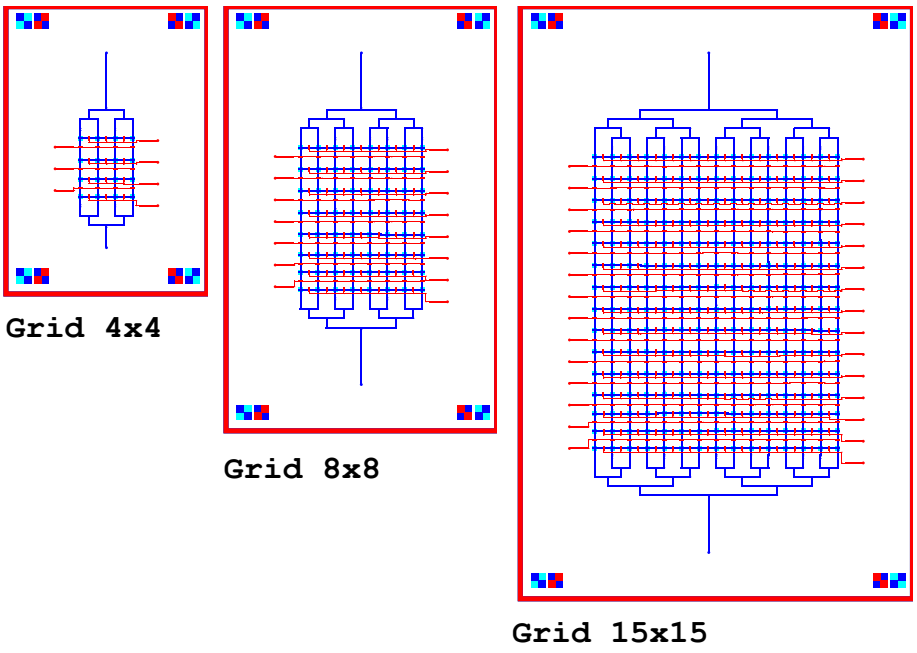


Figure 6·6: Three examples of the grid device (4x4, 8x8, and 15x15) used to test runtimes. The 4x4 grid device contains 96 elements and had a runtime of 49 seconds. The 8x8 grid device contains 376 elements and had a runtime of 7 minutes 16 seconds. Finally, the 15x15 grid device contains 1326 elements and had a runtime of 1 hour 30 minutes and 35 seconds.

in the device increased, I also needed to increase the number of moves per component at each temperature step during simulated annealing to achieve a good placement for routing, which contributes to the increase in runtime. The most computationally intensive portions of the workflow reside in the place-and-route algorithms. Refinement of the placement algorithm to allow swapping component locations and rotation of components as well as using more sophisticated non-grid based routing algorithms may decrease runtime.

6.3 Demonstration of workflow

I next demonstrate the ability to fabricate devices from the designs generated by Fluigi. I ran Fluigi on the netlist for device `mixer-3d` to produce `svg` files for both

Table 6.3: Algorithm runtimes for devices

Name	**Number of Components**	**Number of Channels**	**Nets**	**Total Elements**	**Moves**	**Runtime (hr:min:sec)**	**Device Width (cm)**	**Device Length (cm)**
4x4 Grid	46	46	4	96	250	0:00:49	1.49	2.09
5x5 Grid	71	72	5	148	250	0:01:50	1.59	2.62
6x6 Grid	102	104	6	212	250	0:03:13	1.73	2.79
7x7 Grid	139	142	7	288	250	0:04:48	1.87	2.93
8x8 Grid	182	186	8	376	250	0:07:16	1.98	3.11
9x9 Grid	231	236	9	476	250	0:10:54	2.12	3.39
10x10 Grid	286	292	10	588	250	0:16:31	2.23	3.53
11x11 Grid	345	354	11	710	250	0:20:14	2.37	3.71
12x12 Grid	412	422	12	846	250	0:29:50	2.51	3.71
13x13 Grid	485	496	13	984	300	0:43:36	2.62	4.03
14x14 Grid	564	576	14	1154	400	1:04:53	2.72	4.20
15x15 Grid	649	662	15	1326	500	1:30:35	2.86	4.31
16x16 Grid	740	754	16	1510	750	2:25:21	3.00	4.45

the flow layer and the control layer. These files were processed by the program `otherplan` to produce gcode to control the CNC mill (`othermill ver 2`). Additional information needed that was not provided by Fluigi included the depth of the features on each layer and the placement of the acrylic stock on the spoilboard of the mill. Due to an error in calibration, the gaps in the valves were too small to be machined properly and were left out of the final device. Red and blue dye were injected into the device to show that it could mix the liquids. The device design and the final fabricated device are shown in Figure 6·7.

6.4 Replication of `XOR` experiment

Finally, I went through the process of fabricating and testing a device design by Fluigi using multilayer soft lithography. I chose design `H` (shown in Figure 6·4) as I wanted to test the feasibility of conducting the Voigt `NOR` gate experiment in a microfluidic device. Testing of the device functionality was conducted with colored dye.

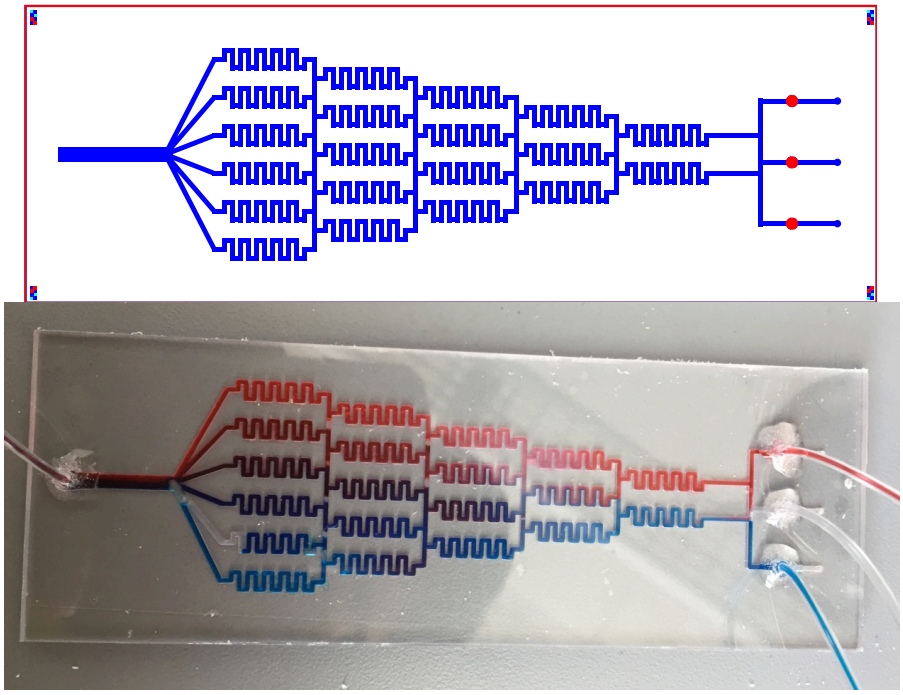


Figure 6·7: 3 input mixer manufactured through CNC milling from a Fluigi generated design. The top image shows the design as generated though Fluigi, and the bottom shows the completed device with red and blue dye mixing in it.

6.4.1 Fabrication process

I sent out the photomask template in `eps` format as generated by Fluigi for device `H` to CADArt Services to be printed on a 8"x10" transparency. The print resolution was 20000dpi, for a minimum feature resolution of 10 μm. Polarity and orientation of the photomask was unchanged from the template file, shown in Figure 6·8.

The full procedure for the photolithography process is explained in Appendix C. The mold for the control layer was made by spinning SU-8 10 at 1000RPM for 60sec, and exposing the photoresist with hard contact for 8 seconds before being developed. The mold for the flow layer was made in two parts, with a different photoresist for each feature height. The height of the cell traps was 3 μm, and the height for the flow channels was 10 μm. For the cell traps, I spun SU-8 2 at 1800RPM for 60sec to

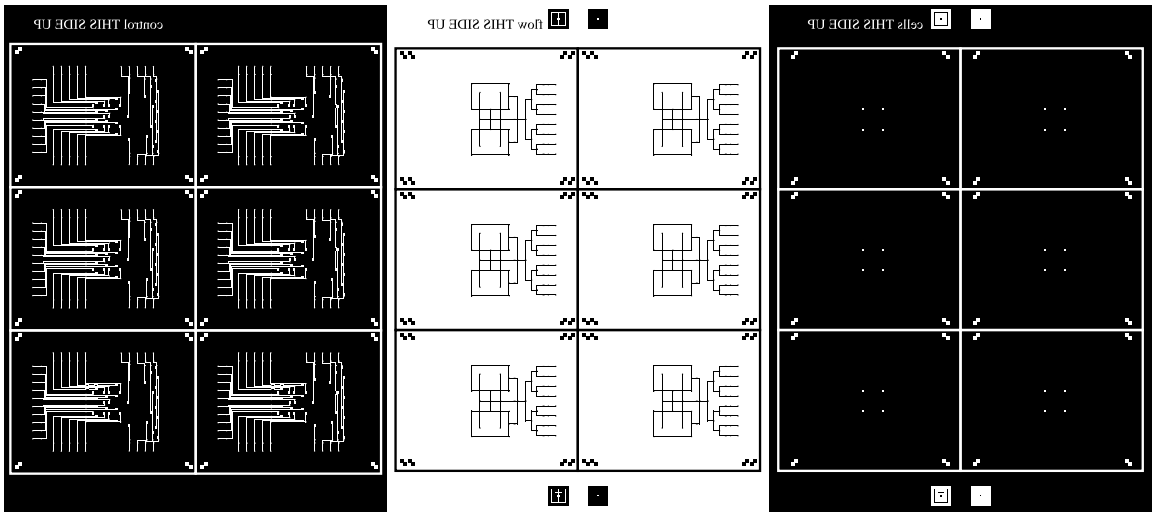


Figure 6·8: Photomask for device `H` used in fabrication. The polarity and orientation of the photomask is identical to the template generated by Fluigi. Six devices can fit onto one 4" wafer. Small alignment marks are in the corner of every device on each mask. The masks for the flow layer and the cell traps show large alignment marks at the top and bottom as they must be aligned on the same wafer.

achieve a $3\mu m$ layer height. The photoresist was exposed for 8 seconds under hard contact and developed for one minute. Then a second photoresist A24629 was spun over the same wafer at 3400RPM for 60sec. The flow channel mask was aligned with the existing pattern on the wafer and exposed for 20 seconds under hard contact. I casted the devices with PDMS, the full procedure being shown in Appendix D, and bonded the completed devices to glass cover slips. The completed device is shown in Figure 6·9.

6.4.2 Device controls

I also used this device to test the ability of Fluigi to generate control patterns given a simple logic function comprised of 4 gates. As the interface to Cello is incomplete, I hard coded a test case of 4 connected biological `NOR` gates (labeled *nor1, nor2, nor3*, and *nor4* with the small signalling molecules *aTc* and *ara* as inputs based on the biological `XOR` gate presented by the Voigt lab (Tamsir et al., 2011). A diagram of

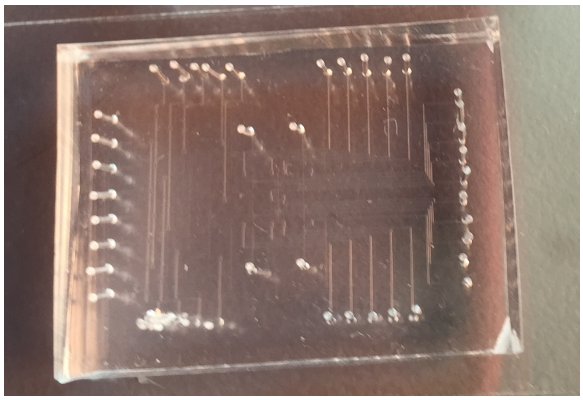


Figure 6·9: Completed 2 layer PDMS device made from the photomask in Figure 6·8. The device has 29 control ports and 12 flow ports.

this function and the breakdown of each logic stage is shown in Figure 6·10A. I used Fluigi to then generate both a microfluidic design that would house the biological circuit and the valve controls needed for circuit functionality. This design, shown in Figure 6·10B, differs slightly from the device `H` in that it has 7 input ports (4 cell inputs, 2 small signal inputs, and 1 media input) where device `H` has 8. The small difference in port numbers will not affect the ability to use these control patterns with device `H` as the extra port will never be accessed by the generated patterns.

The logic function was divided into three stages, as shown in Figure 6·10A. For each stage, Fluigi generated the control patterns for the following actions: applying inputs to the gate(s) either from the input ports or from the previous stage and growing up cells for the current stage. The device is initiated by loading cells into each of the four chambers from the input ports. These controls are output as a set of 4 `Java` files and a `png` image of the device for use with the Biostream GUI and valve controller.

6.4.3 Device testing

I tested the operation of the device with colored dye. I consider the device to be successfully working if liquid can be directed from every chamber to every other

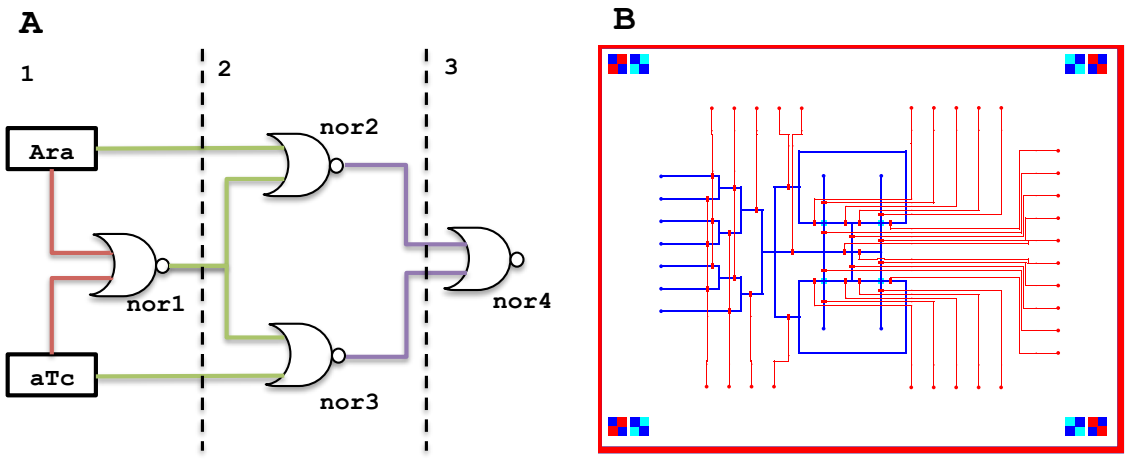


Figure 6·10: The left image shows the biological function used to generate a microfluidic device, and the right image shows the resulting device design. The biological function is divided into three stages of operation, with red lines representing actions in stage 1, green lines in stage 2, and purple lines in stage 3.

chamber. Due to limitations in the experimental setup, I could only actuate 16 valves at a time. As such, I tested the chambers in pairs. For each pair of chambers, I show that fluid can flow to each chamber individually and that fluid can flow between the chambers. I label the chambers and valves as shown in Figure 6·11. The valve patterns for each action is shown in Table 6.4. An "x" represents a closed valve, an "o" represents an open valve, and a "-" represents a valve that can be either state. I loaded chamber 1 with red dye, chamber 2 with yellow dye, chamber 3 with green dye and chamber 4 with blue dye, and ran the valve patterns listed for each path. The results from the test runs are shown in Figure 6·12.

For each path, I first ran dye from the input to the first chamber to the waste port of that chamber. Dye was introduced into the device with a hand-held syringe. I then adjusted the valves such that the flow of the dye was redirected from the first chamber to the second chamber. The valves were controlled through the use of the BUTest Biostream GUI provided by the Khalil lab. The first image of each pair in Figure 6·12 shows the dye flowing to the waste port, and the second image shows the

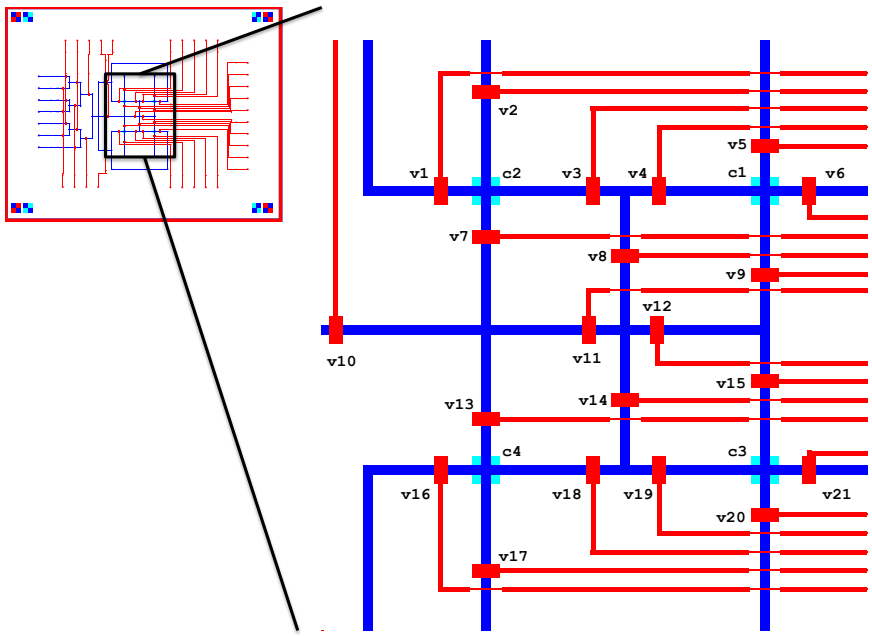


Figure 6·11: The valves and chambers in the test device are numbered as shown. These numbers are referred to in Table 6.4 when determining flow paths between chambers.

Table 6.4: Valve states for fluid paths between chambers

Path	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	v11
c1 to c2	x	o	o	o	x	o	x	x	x	x	-
c1 to c3	-	-	-	x	x	o	-	-	o	x	-
c1 to c4	-	-	-	x	x	o	x	x	o	x	o
c2 to c1	o	x	o	o	o	x	x	x	x	x	-
c2 to c3	o	x	o	x	-	-	x	o	-	x	x
c2 to c4	o	x	x	-	-	-	o	-	-	x	x
c3 to c1	-	-	-	x	o	x	-	-	o	x	-
c3 to c2	x	o	o	x	-	-	x	o	-	x	x
c3 to c4	-	-	-	-	-	-	-	-	-	x	-
c4 to c1	-	-	-	x	o	x	x	x	o	x	o
c4 to c2	x	o	x	-	-	-	o	-	-	x	x
c4 to c3	-	-	-	-	-	-	-	-	-	x	-

Path	v12	v13	v14	v15	v16	v17	v18	v19	v20	v21	
c1 to c2	-	-	-	-	-	-	-	-	-	-	
c1 to c3	x	-	-	o	-	-	-	x	o	x	
c1 to c4	o	o	x	x	x	o	x	-	-	-	
c2 to c1	-	-	-	-	-	-	-	-	-	-	
c2 to c3	x	-	o	x	-	-	x	o	o	x	
c2 to c4	-	o	-	-	x	o	x	-	-	-	
c3 to c1	x	-	-	o	-	-	-	x	x	o	
c3 to c2	x	-	o	x	-	-	x	o	x	o	
c3 to c4	-	x	x	x	x	o	o	o	x	o	
c4 to c1	o	o	x	x	o	x	x	-	-	-	
c4 to c2	-	o	-	-	o	x	x	-	-	-	
c4 to c3	-	x	x	x	o	x	o	o	o	x	

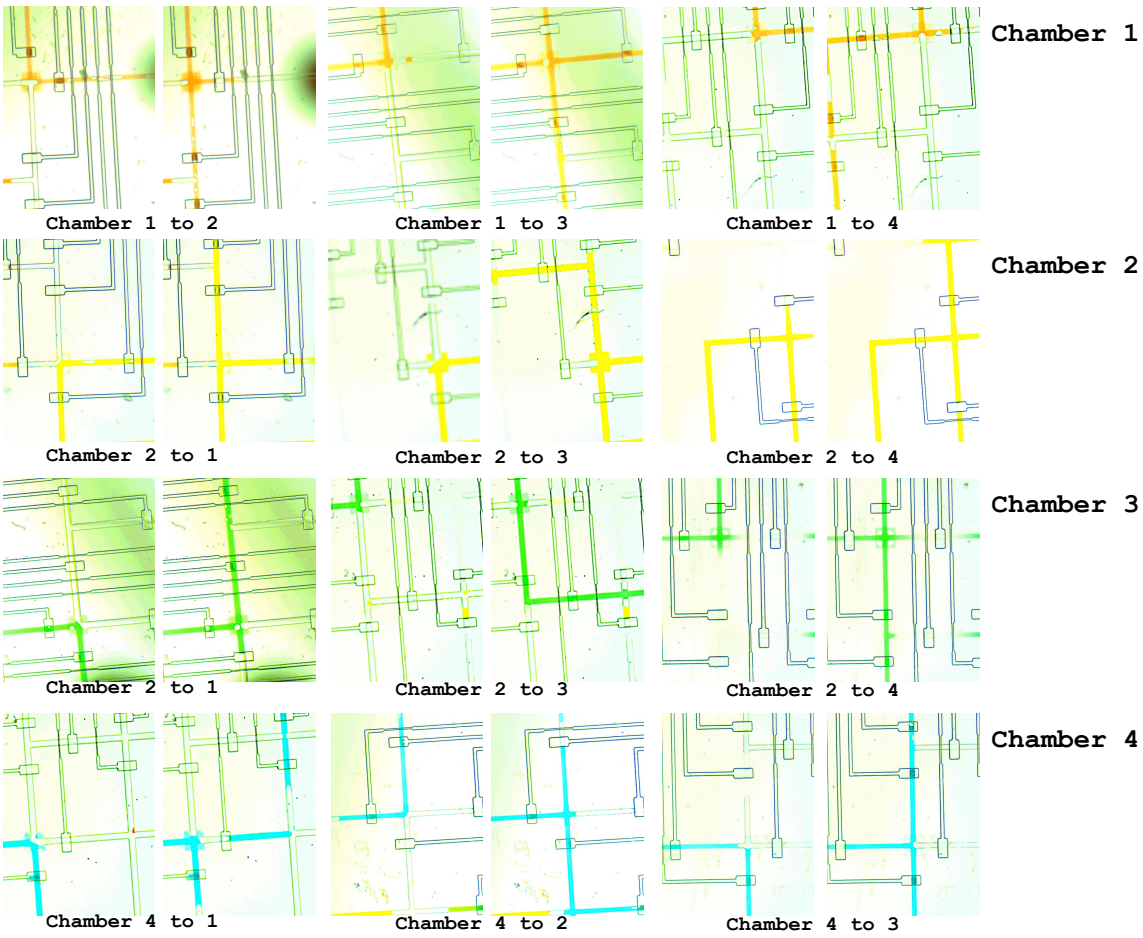


Figure 6·12: Each chamber was loaded with colored dye, and the valves for each flow path were activated. The device was cleared with DI water between runs. The first image in each pair shows the default flow from the input to the chamber to the waste port. The second image shows the flow being directed to a different part of the device.

dye flowing to the second chamber. I flushed the device with water between runs to clear out the dye.

I have shown that I can successfully route fluids between chambers with this device. However, the design must be updated to work reliably with cells or with cell-free tx-tl mix. While fluid can be directed between chambers, the chamber design should be updated to allow the input to mix with the fluid or cells already in the chamber. In addition, the channels to each of the chambers should be balanced for

length, which is not currently the case. For ease of use and to limit potential cracking of the PDMS, the distances between the control ports should be expanded to $2mm$.

Chapter 7

Conclusion and future works

The goal of synthetic biology is to use naturally existing logical constructs in biology (such as repressible and inducible genes) for novel applications in biosensing, therapeutics, and biomaterials. Microfluidics provides reduction in reagent use, increase in high-throughput and automation, and precise control over the spatial and temporal environment necessary increase the scale and robustness of synthetic biology. The integration of microfluidics and synthetic biology has the capability to increase the scale of engineered biological systems for applications in cell-based therapeutics and biosensors, expand on the idea of distributed biological computation, and produce new rapid prototyping platforms for the characterization of genetic devices.

Adoption of microfluidics as an experimental platform in synthetic biology labs has been hindered by the difficulty in designing and fabricating a microfluidic device. Many labs lack both the background in fluid dynamics and the capital equipment needed to fabricate and test microfluidic devices. Design automation tools for microfluidics could reduce the time needed to design and layout devices, and hasten the adoption process.

I have presented here an end-to-end CAD workflow, Fluigi, for designing microfluidic devices. My tool takes as input a microfluidic netlist, and from that netlist, arranges the microfluidic components and connects them in such a way to comply with existing design rules. I used simulated annealing for the placement algorithm and Hadlock's variation on maze routing for the routing algorithm. In addition, I

perform design rule checking on the generated design. My workflow also extends to generating a microfluidic device given a biological function. However, this secondary flow has not been fully integrated into the toolchain. From my workflow, I have fabricated and tested two devices, one made from CNC milling and one made by traditional multilayer soft lithography.

My long-term goal is to develop Fluigi into a fully integrated hardware-software platform for designing and operating microfluidic devices. I show in Figure 7·1 the main blocks of that workflow. Fluigi will incorporate electronic sensors and heaters as part of the device design process, and integrate those controls with the valve controller. There will be real-time feedback from the sensors to the valve controller to close the loop on experimental control and allow dynamic reconfiguration of the microfluidic device. Fluigi will provide the CAD workflow of place-and-route and design rule checking that I have previously described and provide a GUI for users who prefer graphical to textual inputs. Ideally, the inputs will be expanded to include a language for describing experiments, and from that description, Fluigi will synthesize a microfluidic device and the controls to operate the device. To reach that goal, the following improvements should be made to the existing architecture and software workflow.

The current netlist input format and underlying device description should be updated to expand the flexibility of Fluigi. The main flaw in the current netlist format is the inability to define exact spacing between components, including specifying channel lengths as well as widths. One potential solution is to generate rules about the positions of components and their relationships, and evaluate those rules on the device as part of the place-and-route flow. In addition, the netlist grammar should be expanded to include defining modules in the netlist instead of relying on pre-built and pre-defined structures. The netlist should also be expanded to take into account

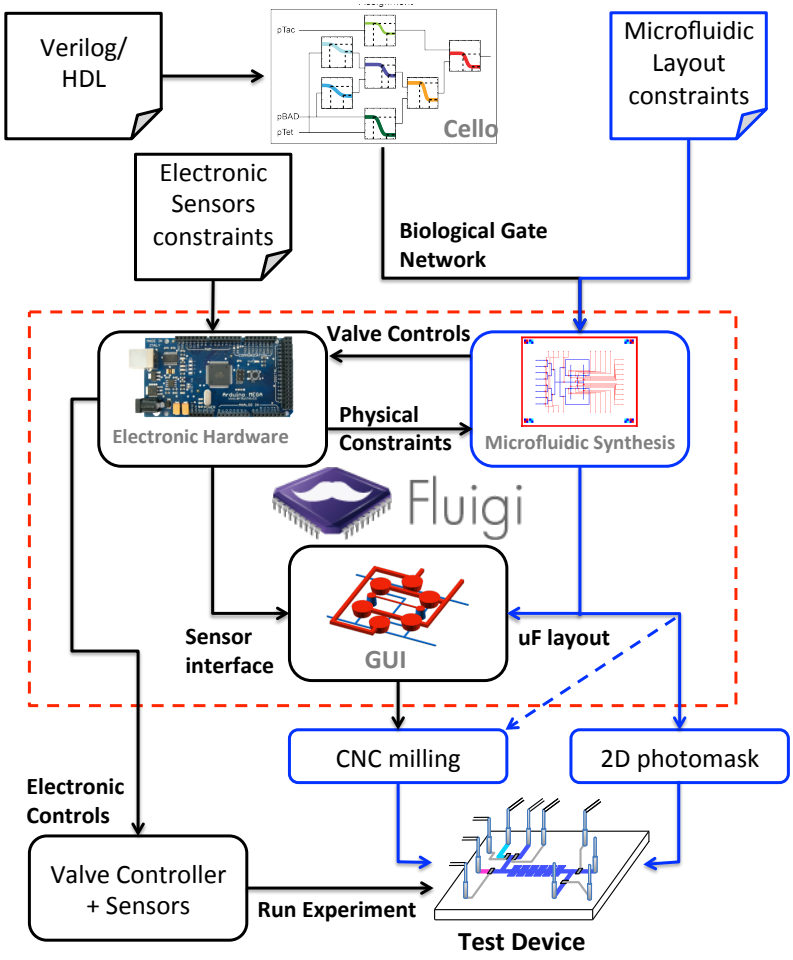


Figure 7·1: The longterm workflow of Fluigi, which includes hardware control and chip interfaces, microfluidic CAD, and rendering and fabricating of molds for microfluidic devices. Highlighted in blue are sections that have been completed, which includes synthesis of a microfluidic layout from design constraints and an input netlist through device fabrication. Future work will integrate a GUI for module design, layout constraints for electronic sensors, and accept biological gate networks as inputs.

more fabrication technologies, and with that, allow the definition of diagonal channels and more primitives such as generic round and rectangular chambers. Ultimately, I would like to describe experiments, and infer device architecture from the description.

More sophisticated algorithms are necessary for Fluigi to handle larger and more complex device structures. A partition and floorplanning step should be introduced prior to placement. The placement algorithms should be updated to include swaps

between components and rotation of components to achieve better layouts. Previously rejected layouts should be remembered and tracked to find better final placements. I may also wish to investigate alternative placement algorithms such as force directed placement.

The routing algorithm needs to be able to incorporate the ability to route diagonal channels. To ensure equal flow rates into the device, flow channel length balancing should be introduced as part of design rule checking. Finally, compaction algorithms should be implemented to reduce the amount of white space in device designs while retaining some amount of buffer necessary to accomodate human error in the fabrication process.

My results show that an end-to-end workflow for microfluidic design for synthetic biology is possible. The integration of microfluidics and synthetic biology has the capability to increase the scale of engineered biological systems for applications in DNA assembly, biosensors, and screening assays for novel orthogonal genetic parts. Fluigi and the functions it provides represent an important step towards this integration.

Appendix A

Java libraries

Table A.1: Additional Java libraries

Library	Description	Location
jgrapht	graph library	http://jgrapht.org/
antlr-3.5.2	ANTLR parser/lexer	http://www.antlr3.org/
epsgraphics-1.4	output eps formal	http://www.abeel.be/wiki/EPSGraphics
jfreesvg-3.0	output svg format	http://www.jfree.org/jfreesvg/
algs4	utilities for design rule checking	http://algs4.cs.princeton.edu/code/

Appendix B

Device Netlists

B.1 Device A

```
DEVICE flow_focus

LAYER FLOW
V DROPLET GENERATOR FLOW FOCUS ff radius=100 oilChannelWidth=100
    waterChannelWidth=40 angle=30 length=500;

PORT p r=100;
NODE n1;
NODE n2;

CHANNEL c1 from ff 1 to n1 1 w=100;
CHANNEL c2 from n1 3 to n2 1 w=20;
CHANNEL c3 from n2 3 to p 1 w=100;

END LAYER
```

B.2 Device B

```
DEVICE simple

LAYER FLOW
PORT p1, p2, p3 r=100;
NODE n1;
H LONG CELL TRAP ct1 numChambers=10 chamberWidth=100 chamberLength=100
    chamberSpacing=50 channelWidth=100 ;

CHANNEL c1 from p1 3 to n1 1 w=100;
```

```
CHANNEL c2 from p2 1 to n1 3 w=100;
CHANNEL c3 from n1 2 to ct1 1 w=100;
CHANNEL c4 from ct1 2 to p3 4 w=100;

END LAYER

LAYER CONTROL
PORT cp1, cp2 r=100;
VALVE v1 on c1 w=300 l=100;
VALVE v2 on c2 w=300 l=100;
CHANNEL c5 from cp1 2 to v1 4 w=50;
CHANNEL c6 from cp2 2 to v2 4 w=50;

END LAYER
```

B.3 Device C

```
DEVICE tdroplet

LAYER FLOW
H DROPLET GENERATOR T t1 radius=100 oilChannelWidth=100 waterChannelWidth
    =20;
H MIXER x1 numBends=5 bendSpacing=50 bendLength=1000 channelWidth=100;
H DROPLET GENERATOR T t2 radius=100 oilChannelWidth=100 waterChannelWidth
    =20;
H MIXER x2 numBends=5 bendSpacing=50 bendLength=1000 channelWidth=100;
V MUX m1 2 to 1 spacing=4000 flowChannelWidth=100 controlChannelWidth=50;
H LONG CELL TRAP ct1 numChambers=10 chamberWidth=100 chamberLength=100
    chamberSpacing=50 channelWidth=100 ;
PORT p1 r=100;

CHANNEL c1 from t1 1 to x1 1 w=100;
CHANNEL c2 from x1 2 to m1 1 w=100;
CHANNEL c3 from t2 1 to x2 1 w=100;
CHANNEL c4 from x2 2 to m1 2 w=100;
CHANNEL c5 from m1 3 to ct1 1 w=100;
CHANNEL c6 from ct1 2 to p1 4 w=100;
```

```
END LAYER

LAYER CONTROL
PORT cp1, cp2 r=100;

CHANNEL cc1 from cp1 3 to m1 4 w=100;
CHANNEL cc2 from cp2 1 to m1 5 w=100;
END LAYER
```

B.4 Device D

```
DEVICE grad_cells

LAYER FLOW
V BANK pb1 of 2 PORT r=100 dir=RIGHT spacing=1200 channelWidth=100;
PORT p1 r=100;
NODE n1;
H GRADIENT GENERATOR g 1 to 4 numBends=10 bendSpacing=100 bendLength=500
    channelWidth=100;
V MUX m1 1 to 4 spacing=500 flowChannelWidth=100 controlChannelWidth=50;
V TREE t1 4 to 1 spacing=500 flowChannelWidth=100;
V BANK ctb of 4 CELL TRAP numChambers=20 chamberWidth=100 chamberLength=100
     chamberSpacing=30 spacing=500 channelWidth=100;

CHANNEL c1 from pb1 1 to n1 1 w=100;
CHANNEL c2 from pb1 2 to n1 3 w=100;
CHANNEL c3 from n1 2 to g 1 w=100;
CHANNEL c4 from g 2 to m1 1 w=100;
CHANNEL c5 from m1 2 to ctb 1 w=100;
CHANNEL c6 from m1 3 to ctb 2 w=100;
CHANNEL c7 from m1 4 to ctb 3 w=100;
CHANNEL c8 from m1 5 to ctb 4 w=100;
CHANNEL c9 from ctb 5 to t1 1 w=100;
CHANNEL c10 from ctb 6 to t1 2 w=100;
CHANNEL c11 from ctb 7 to t1 3 w=100;
CHANNEL c12 from ctb 8 to t1 4 w=100;
CHANNEL c13 from t1 5 to p1 4 w=100;
```

```
END LAYER

LAYER CONTROL
H BANK cpb1 of 2 PORT r=100 dir=DOWN spacing=1200 channelWidth=50;
H BANK cpb2 of 2 PORT r=100 dir=UP spacing=1200 channelWidth=50;
CHANNEL cc1 from cpb1 1 to m1 7 w=50;
CHANNEL cc2 from cpb1 2 to m1 9 w=50;
CHANNEL cc3 from cpb2 1 to m1 6 w=50;
CHANNEL cc4 from cpb2 2 to m1 8 w=50;
END LAYER
```

B.5 Device E

```
DEVICE multi_input

LAYER FLOW
H BANK pb1 of 2 PORT r=100 dir=DOWN spacing=1200 channelWidth=100;
V BANK pb2 of 2 PORT r=100 dir=RIGHT spacing=1200 channelWidth=100;
V BANK pb3 of 2 PORT r=100 dir=LEFT spacing=1200 channelWidth=100;
H MUX m1 2 to 1 spacing=1200 flowChannelWidth=100 controlChannelWidth=50;
V TREE t1 2 to 1 spacing=1200 flowChannelWidth=100;
V TREE t2 1 to 2 spacing=1200 flowChannelWidth=100;
H MIXER x1 numBends=10 bendSpacing=100 bendLength=1000 channelWidth=100;
H MIXER x2 numBends=10 bendSpacing=100 bendLength=1000 channelWidth=100;
PORT p1 r=100;
SQUARE CELL TRAP ct1 chamberWidth=500 chamberLength=500 channelWidth=100;

CHANNEL c1 from pb1 1 to m1 1 w=100;
CHANNEL c2 from pb1 2 to m1 2 w=100;
CHANNEL c3 from m1 3 to ct1 1 w=100;

CHANNEL c4 from pb2 1 to t1 1 w=100;
CHANNEL c5 from pb2 2 to t1 2 w=100;
CHANNEL c6 from t1 3 to x1 1 w=100;
CHANNEL c7 from x1 2 to ct1 4 w=100;

CHANNEL c8 from pb3 1 to t2 2 w=100;
CHANNEL c9 from pb3 2 to t2 3 w=100;
```

```
CHANNEL c10 from t2 1 to x2 2 w=100;
CHANNEL c11 from x2 1 to ct1 2 w=100;

CHANNEL c12 from ct1 3 to p1 1 w=100;
END LAYER

LAYER CONTROL
PORT cp1, cp2, cp3, cp4 r=100;

CHANNEL cc1 from cp1 3 to m1 4 w=50;
CHANNEL cc2 from cp2 3 to m1 5 w=50;

VALVE v1 on c7 w=150 l=300;
VALVE v2 on c11 w=150 l=300;

CHANNEL cc3 from cp3 2 to v1 3 w=50;
CHANNEL cc4 from cp4 4 to v2 3 w=50;
END LAYER
```

B.6 Device F

```
DEVICE hasty

LAYER FLOW
PORT p1, p2, p3, p4, p5 r=100;
NODE n1, n2;
V TREE t1 1 to 8 spacing=500 flowChannelWidth=100;
V TREE t2 8 to 1 spacing=500 flowChannelWidth=100;
V BANK b1 of 8 CELL TRAP numChambers=100 chamberWidth=100 chamberLength=100
    chamberSpacing=30 spacing=500 channelWidth=100;

CHANNEL c1 from p1 3 to n1 1 w=100;
CHANNEL c2 from p2 2 to n1 4 w=100;
CHANNEL c3 from p3 1 to n1 3 w=100;
CHANNEL c4 from n1 2 to t1 1 w=100;

CHANNEL c5 from t1 2 to b1 1 w=100;
CHANNEL c6 from t1 3 to b1 2 w=100;
```

```
CHANNEL c7 from t1 4 to b1 3 w=100;
CHANNEL c8 from t1 5 to b1 4 w=100;
CHANNEL c9 from t1 6 to b1 5 w=100;
CHANNEL c10 from t1 7 to b1 6 w=100;
CHANNEL c11 from t1 8 to b1 7 w=100;
CHANNEL c12 from t1 9 to b1 8 w=100;

CHANNEL c13 from t2 1 to b1 9 w=100;
CHANNEL c14 from t2 2 to b1 10 w=100;
CHANNEL c15 from t2 3 to b1 11 w=100;
CHANNEL c16 from t2 4 to b1 12 w=100;
CHANNEL c17 from t2 5 to b1 13 w=100;
CHANNEL c18 from t2 6 to b1 14 w=100;
CHANNEL c19 from t2 7 to b1 15 w=100;
CHANNEL c20 from t2 8 to b1 16 w=100;

CHANNEL c21 from t2 9 to n2 4 w=100;
CHANNEL c22 from n2 1 to p4 3 w=100;
CHANNEL c23 from n2 3 to p5 1 w=100;
END LAYER
```

B.7 Device G

```
DEVICE rotary_cells

LAYER FLOW
V MUX m1 4 to 1 spacing=1200 flowChannelWidth=100 controlChannelWidth=50;
V MUX m2 1 to 4 spacing=500 flowChannelWidth=100 controlChannelWidth=50;
V BANK pb1 of 4 PORT r=100 dir=RIGHT spacing=1200 channelWidth=100;
H ROTARY PUMP rp radius=1000 flowChannelWidth=100 controlChannelWidth=50;
V BANK b1 of 4 CELL TRAP numChambers=100 chamberWidth=100 chamberLength=100
    chamberSpacing=30 spacing=500 channelWidth=100;
V TREE t1 4 to 1 spacing=500 flowChannelWidth=100;
PORT p1 r=100;

CHANNEL c1 from pb1 1 to m1 1 w=100;
CHANNEL c2 from pb1 2 to m1 2 w=100;
CHANNEL c3 from pb1 3 to m1 3 w=100;
```

```
CHANNEL c4 from pb1 4 to m1 4 w=100;
CHANNEL c5 from m1 5 to rp 1 w=100;
CHANNEL c6 from rp 2 to m2 1 w=100;
CHANNEL c7 from m2 2 to b1 1 w=100;
CHANNEL c8 from m2 3 to b1 2 w=100;
CHANNEL c9 from m2 4 to b1 3 w=100;
CHANNEL c10 from m2 5 to b1 4 w=100;
CHANNEL c11 from b1 5 to t1 1 w=100;
CHANNEL c12 from b1 6 to t1 2 w=100;
CHANNEL c13 from b1 7 to t1 3 w=100;
CHANNEL c14 from b1 8 to t1 4 w=100;
CHANNEL c15 from t1 5 to p1 4 w=100;

END LAYER

LAYER CONTROL
H BANK cb1 of 2 PORT r=100 dir=DOWN spacing=1200 channelWidth=50;
H BANK cb2 of 2 PORT r=100 dir=UP spacing=1200 channelWidth=50;
H BANK cb3 of 3 PORT r=100 dir=DOWN spacing=1200 channelWidth=50;
H BANK cb4 of 2 PORT r=100 dir=UP spacing=1200 channelWidth=50;
H BANK cb5 of 2 PORT r=100 dir=DOWN spacing=1200 channelWidth=50;
H BANK cb6 of 2 PORT r=100 dir=UP spacing=1200 channelWidth=50;

CHANNEL cc1 from m1 6 to cb1 1 w=50;
CHANNEL cc2 from m1 8 to cb1 2 w=50;
CHANNEL cc3 from rp 3 to cb3 1 w=50;
CHANNEL cc4 from rp 6 to cb3 2 w=50;
CHANNEL cc5 from rp 7 to cb3 3 w=50;
CHANNEL cc6 from m2 7 to cb5 1 w=50;
CHANNEL cc7 from m2 9 to cb5 2 w=50;
CHANNEL cc8 from m1 7 to cb2 1 w=50;
CHANNEL cc9 from m1 9 to cb2 2 w=50;
CHANNEL cc10 from rp 4 to cb4 1 w=50;
CHANNEL cc11 from rp 5 to cb4 2 w=50;
CHANNEL cc12 from m2 6 to cb6 1 w=50;
CHANNEL cc13 from m2 8 to cb6 2 w=50;
```

```
END LAYER
```

B.8 Device H

```
DEVICE logic04

LAYER FLOW

LOGIC ARRAY la flowChannelWidth=100 controlChannelWidth=50 chamberLength
    =100 chamberWidth=100 r=100;
V BANK b0 of 8 PORT r=100 dir=RIGHT spacing=1500 channelWidth=100;
V MUX m1 8 to 1 spacing=1500 flowChannelWidth=100 controlChannelWidth=50;

NODE n1;

CHANNEL c0 from m1 9 to n1 4 w=100;
CHANNEL c1 from n1 2 to la 3 w=100;
CHANNEL c2 from n1 3 to la 2 w=100;
CHANNEL c3 from la 1 to n1 1 w=100;
CHANNEL c4 from b0 1 to m1 1 w=100;
CHANNEL c5 from b0 2 to m1 2 w=100;
CHANNEL c6 from b0 3 to m1 3 w=100;
CHANNEL c7 from b0 4 to m1 4 w=100;
CHANNEL c8 from b0 5 to m1 5 w=100;
CHANNEL c9 from b0 6 to m1 6 w=100;
CHANNEL c10 from b0 7 to m1 7 w=100;
CHANNEL c11 from b0 8 to m1 8 w=100;

END LAYER

LAYER CONTROL
H BANK b4 of 5 PORT r=100 dir=DOWN spacing=1200 channelWidth=50;
H BANK b5 of 4 PORT r=100 dir=UP spacing=1200 channelWidth=50;

CHANNEL cc21 from m1 10 to b4 1 w=50;
CHANNEL cc22 from m1 11 to b5 1 w=50;
CHANNEL cc23 from m1 12 to b4 2 w=50;
CHANNEL cc24 from m1 13 to b5 2 w=50;
```

```
CHANNEL cc25 from m1 14 to b4 3w=50;
CHANNEL cc26 from m1 15 to b5 3 w=50;

CHANNEL cca from la 24 to b4 4 w=50;
CHANNEL ccb from la 25 to b4 5 w=50;
CHANNEL ccc from la 26 to b5 4 w=50;


H BANK b1 of 5 PORT r=100 dir=DOWN spacing=1200 channelWidth=50;
H BANK b3 of 5 PORT r=100 dir=UP spacing=1200 channelWidth=50;
V BANK b2 of 10 PORT r=100 dir=LEFT spacing=1200 channelWidth=50;

CHANNEL cc10 from la 13 to b2 5 w=50;
CHANNEL cc11 from la 14 to b2 6 w=50;
CHANNEL cc9 from la 12 to b2 4 w=50;
CHANNEL cc12 from la 15 to b2 7 w=50;
CHANNEL cc8 from la 11 to b2 3 w=50;
CHANNEL cc13 from la 16 to b2 8 w=50;
CHANNEL cc7 from la 10 to b2 2 w=50;
CHANNEL cc14 from la 17 to b2 9 w=50;
CHANNEL cc6 from la 9 to b2 1 w=50;
CHANNEL cc15 from la 18 to b2 10 w=50;

CHANNEL cc1 from b1 1 to la 4 w=50;
CHANNEL cc5 from b1 5 to la 8 w=50;
CHANNEL cc2 from b1 2 to la 5 w=50;
CHANNEL cc4 from b1 4 to la 7 w=50;
CHANNEL cc3 from b1 3 to la 6 w=50;


CHANNEL cc16 from b3 1 to la 19 w=50;
CHANNEL cc17 from b3 2 to la 20 w=50;
CHANNEL cc18 from b3 3 to la 21 w=50;
CHANNEL cc19 from b3 4 to la 22 w=50;
CHANNEL cc20 from b3 5 to la 23 w=50;
END LAYER
```

B.9 Device I

```
DEVICE rotary16

LAYER FLOW
H BANK pb1 of 16 PORT r=100 dir=DOWN spacing=1200 channelWidth=100;
H MUX m1 16 to 1 spacing=1200 flowChannelWidth=100 controlChannelWidth=50;
V ROTARY PUMP rp radius=1000 flowChannelWidth=100 controlChannelWidth=50;
H BANK pb2 of 16 PORT r=100 dir=UP spacing=1200 channelWidth=100;
H MUX m2 1 to 16 spacing=1200 flowChannelWidth=100 controlChannelWidth=50;

CHANNEL c1 from pb1 1 to m1 1 w=100;
CHANNEL c2 from pb1 2 to m1 2 w=100;
CHANNEL c3 from pb1 3 to m1 3 w=100;
CHANNEL c4 from pb1 4 to m1 4 w=100;
CHANNEL c5 from pb1 5 to m1 5 w=100;
CHANNEL c6 from pb1 6 to m1 6 w=100;
CHANNEL c7 from pb1 7 to m1 7 w=100;
CHANNEL c8 from pb1 8 to m1 8 w=100;
CHANNEL c9 from pb1 9 to m1 9 w=100;
CHANNEL c10 from pb1 10 to m1 10 w=100;
CHANNEL c11 from pb1 11 to m1 11 w=100;
CHANNEL c12 from pb1 12 to m1 12 w=100;
CHANNEL c13 from pb1 13 to m1 13 w=100;
CHANNEL c14 from pb1 14 to m1 14 w=100;
CHANNEL c15 from pb1 15 to m1 15 w=100;
CHANNEL c16 from pb1 16 to m1 16 w=100;

CHANNEL c17 from m1 17 to rp 1 w=100;
CHANNEL c18 from rp 2 to m2 1 w=100;

CHANNEL c19 from m2 2 to pb2 1 w=100;
CHANNEL c20 from m2 3 to pb2 2 w=100;
CHANNEL c21 from m2 4 to pb2 3 w=100;
CHANNEL c22 from m2 5 to pb2 4 w=100;
CHANNEL c23 from m2 6 to pb2 5 w=100;
CHANNEL c24 from m2 7 to pb2 6 w=100;
```

```
CHANNEL c25 from m2 8 to pb2 7 w=100;
CHANNEL c26 from m2 9 to pb2 8 w=100;
CHANNEL c27 from m2 10 to pb2 9 w=100;
CHANNEL c28 from m2 11 to pb2 10 w=100;
CHANNEL c29 from m2 12 to pb2 11 w=100;
CHANNEL c30 from m2 13 to pb2 12 w=100;
CHANNEL c31 from m2 14 to pb2 13 w=100;
CHANNEL c32 from m2 15 to pb2 14 w=100;
CHANNEL c33 from m2 16 to pb2 15 w=100;
CHANNEL c34 from m2 17 to pb2 16 w=100;
END LAYER

LAYER CONTROL
V BANK cpb1 of 4 PORT r=100 dir=RIGHT spacing=1200 channelWidth=100;
V BANK cpb2 of 4 PORT r=100 dir=LEFT spacing=1200 channelWidth=100;
V BANK cpb3 of 2 PORT r=100 dir=RIGHT spacing=1200 channelWidth=100;
V BANK cpb4 of 3 PORT r=100 dir=LEFT spacing=1200 channelWidth=100;
V BANK cpb5 of 4 PORT r=100 dir=RIGHT spacing=1200 channelWidth=100;
V BANK cpb6 of 4 PORT r=100 dir=LEFT spacing=1200 channelWidth=100;

CHANNEL cc1 from m1 18 to cpb1 1 w=50;
CHANNEL cc2 from m1 20 to cpb1 2 w=50;
CHANNEL cc3 from m1 22 to cpb1 3 w=50;
CHANNEL cc4 from m1 24 to cpb1 4 w=50;
CHANNEL cc5 from m1 19 to cpb2 1 w=50;
CHANNEL cc6 from m1 21 to cpb2 2 w=50;
CHANNEL cc7 from m1 23 to cpb2 3 w=50;
CHANNEL cc8 from m1 25 to cpb2 4 w=50;
CHANNEL cc9 from m2 18 to cpb6 1 w=50;
CHANNEL cc10 from m2 20 to cpb6 2 w=50;
CHANNEL cc11 from m2 22 to cpb6 3 w=50;
CHANNEL cc12 from m2 24 to cpb6 4 w=50;
CHANNEL cc13 from m2 19 to cpb5 1 w=50;
CHANNEL cc14 from m2 21 to cpb5 2 w=50;
CHANNEL cc15 from m2 23 to cpb5 3 w=50;
CHANNEL cc16 from m2 25 to cpb5 4 w=50;
```

```
CHANNEL cc17 from rp 4 to cpb3 1 w=50;
CHANNEL cc18 from rp 5 to cpb3 2 w=50;
CHANNEL cc19 from rp 3 to cpb4 1 w=50;
CHANNEL cc20 from rp 6 to cpb4 2 w=50;
CHANNEL cc21 from rp 7 to cpb4 3 w=50;
END LAYER
```

B.10 Device J

```
DEVICE net_mux

LAYER FLOW
V BANK b1 of 8 PORT r=100 dir=RIGHT spacing=1500 channelWidth=100;
V BANK b2 of 8 CELL TRAP numChambers=10 chamberWidth=100 chamberLength=100
    chamberSpacing=30 spacing=1500 channelWidth=100;
V TREE m1 8 to 1 spacing=1500 flowChannelWidth=100;
H MUX m2 2 to 1 spacing=1500 flowChannelWidth=100 controlChannelWidth=50;
H BANK b3 of 2 PORT r=100 dir=DOWN spacing=1500 channelWidth=100;
PORT p1, p2 r=100;
NODE n1, n2, n3, n4, n5, n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16;

CHANNEL c1 from b1 1 to n1 4 w=100;
CHANNEL c2 from b1 2 to n2 4 w=100;
CHANNEL c3 from b1 3 to n3 4 w=100;
CHANNEL c4 from b1 4 to n4 4 w=100;
CHANNEL c5 from b1 5 to n5 4 w=100;
CHANNEL c6 from b1 6 to n6 4 w=100;
CHANNEL c7 from b1 7 to n7 4 w=100;
CHANNEL c8 from b1 8 to n8 4 w=100;

CHANNEL c9 from n1 3 to n2 1 w=100;
CHANNEL c10 from n2 3 to n3 1 w=100;
CHANNEL c11 from n3 3 to n4 1 w=100;
CHANNEL c12 from n4 3 to n5 1 w=100;
CHANNEL c13 from n5 3 to n6 1 w=100;
CHANNEL c14 from n6 3 to n7 1 w=100;
CHANNEL c15 from n7 3 to n8 1 w=100;
```

```
CHANNEL c16 from n1 2 to b2 1 w=100;
CHANNEL c17 from n2 2 to b2 2 w=100;
CHANNEL c18 from n3 2 to b2 3 w=100;
CHANNEL c19 from n4 2 to b2 4 w=100;
CHANNEL c20 from n5 2 to b2 5 w=100;
CHANNEL c21 from n6 2 to b2 6 w=100;
CHANNEL c22 from n7 2 to b2 7 w=100;
CHANNEL c23 from n8 2 to b2 8 w=100;

CHANNEL c24 from b2 9 to n9 4 w=100;
CHANNEL c25 from b2 10 to n10 4 w=100;
CHANNEL c26 from b2 11 to n11 4 w=100;
CHANNEL c27 from b2 12 to n12 4 w=100;
CHANNEL c28 from b2 13 to n13 4 w=100;
CHANNEL c29 from b2 14 to n14 4 w=100;
CHANNEL c30 from b2 15 to n15 4 w=100;
CHANNEL c31 from b2 16 to n16 4 w=100;

CHANNEL c32 from n9 3 to n10 1 w=100;
CHANNEL c33 from n10 3 to n11 1 w=100;
CHANNEL c34 from n11 3 to n12 1 w=100;
CHANNEL c35 from n12 3 to n13 1 w=100;
CHANNEL c36 from n13 3 to n14 1 w=100;
CHANNEL c37 from n14 3 to n15 1 w=100;
CHANNEL c38 from n15 3 to n16 1 w=100;
CHANNEL c39 from n16 3 to p1 1 w=100;

CHANNEL c40 from n9 2 to m1 1 w=100;
CHANNEL c41 from n10 2 to m1 2 w=100;
CHANNEL c42 from n11 2 to m1 3 w=100;
CHANNEL c43 from n12 2 to m1 4 w=100;
CHANNEL c44 from n13 2 to m1 5 w=100;
CHANNEL c45 from n14 2 to m1 6 w=100;
CHANNEL c46 from n15 2 to m1 7 w=100;
CHANNEL c47 from n16 2 to m1 8 w=100;

CHANNEL c48 from m1 9 to p2 4 w=100;
```

```
CHANNEL c49 from n1 1 to m2 3 w=100;
CHANNEL c50 from b3 1 to m2 1 w=100;
CHANNEL c51 from b3 2 to m2 2 w=100;

END LAYER

LAYER CONTROL
PORT cp1, cp2, cp3, cp4, cp5, cp6 r=100;

CHANNEL cm1 from cp1 2 to m2 4 w=50;
CHANNEL cm2 from cp2 3 to m2 5 w=50;

VALVE v1 on c1 w=150 l=300;
VALVE v2 on c2 w=150 l=300;
VALVE v3 on c3 w=150 l=300;
VALVE v4 on c4 w=150 l=300;
VALVE v5 on c5 w=150 l=300;
VALVE v6 on c6 w=150 l=300;
VALVE v7 on c7 w=150 l=300;
VALVE v8 on c8 w=150 l=300;
CHANNEL cc1 from v1 3 to v2 1 w=50;
CHANNEL cc2 from v2 3 to v3 1 w=50;
CHANNEL cc3 from v3 3 to v4 1 w=50;
CHANNEL cc4 from v4 3 to v5 1 w=50;
CHANNEL cc5 from v5 3 to v6 1 w=50;
CHANNEL cc6 from v6 3 to v7 1 w=50;
CHANNEL cc7 from v7 3 to v8 1 w=50;
CHANNEL cc8 from v8 3 to cp3 1 w=50;

VALVE v9 on c49 w=300 l=150;
VALVE v10 on c9 w=300 l=150;
VALVE v11 on c10 w=300 l=150;
VALVE v12 on c11 w=300 l=150;
VALVE v13 on c12 w=300 l=150;
VALVE v14 on c13 w=300 l=150;
VALVE v15 on c14 w=300 l=150;
```

```
VALVE v16 on c15 w=300 l=150;
NET n1 from cp4 1 to v16 4, v10 4, v11 4, v12 4, v13 4, v14 4, v15 4, v9 4
    channelWidth=50;

VALVE v17 on c32 w=300 l=150;
VALVE v18 on c33 w=300 l=150;
VALVE v19 on c34 w=300 l=150;
VALVE v20 on c35 w=300 l=150;
VALVE v21 on c36 w=300 l=150;
VALVE v22 on c37 w=300 l=150;
VALVE v23 on c38 w=300 l=150;
VALVE v24 on c39 w=300 l=150;


VALVE v25 on c40 w=150 l=300;
VALVE v26 on c41 w=150 l=300;
VALVE v27 on c42 w=150 l=300;
VALVE v28 on c43 w=150 l=300;
VALVE v29 on c44 w=150 l=300;
VALVE v30 on c45 w=150 l=300;
VALVE v31 on c46 w=150 l=300;
VALVE v32 on c47 w=150 l=300;
CHANNEL cc41 from v25 3 to v26 1 w=50;
CHANNEL cc42 from v26 3 to v27 1 w=50;
CHANNEL cc43 from v27 3 to v28 1 w=50;
CHANNEL cc44 from v28 3 to v29 1 w=50;
CHANNEL cc45 from v29 3 to v30 1 w=50;
CHANNEL cc46 from v30 3 to v31 1 w=50;
CHANNEL cc47 from v31 3 to v32 1 w=50;
CHANNEL cc48 from v25 1 to cp6 4 w=50;
NET n2 from cp5 1 to v24 2, v18 2, v19 2, v20 2, v21 2, v22 2, v23 2, v17 2
     channelWidth=50;

END LAYER
```

B.11 Device K

```
DEVICE grid_8
```

```
LAYER FLOW
H BANK b1 of 8 PORT r=100 dir=DOWN spacing=1200 channelWidth=100;
H MUX m0 8 to 1 spacing=1200 flowChannelWidth=100 controlChannelWidth=50;
H TREE m1 1 to 8 spacing=1200 flowChannelWidth=100;
H TREE m2 8 to 1 spacing=1200 flowChannelWidth=100;
PORT p2 r=100;
SQUARE CELL TRAP ct1, ct2, ct3, ct4, ct5, ct6, ct7, ct8, ct9, ct10, ct11,
    ct12, ct13, ct14, ct15, ct16, ct17, ct18, ct19, ct20, ct21, ct22, ct23,
     ct24, ct25, ct26, ct27, ct28, ct29, ct30, ct31, ct32, ct33, ct34, ct35
    , ct36, ct37, ct38, ct39, ct40, ct41, ct42, ct43, ct44, ct45, ct46,
    ct47, ct48, ct49, ct50, ct51, ct52, ct53, ct54, ct55, ct56, ct57, ct58,
     ct59, ct60, ct61, ct62, ct63, ct64 chamberWidth=100 chamberLength=100
    channelWidth=100;
CHANNEL ca from b1 1 to m0 1 w=100;
CHANNEL cb from b1 2 to m0 2 w=100;
CHANNEL cc from b1 3 to m0 3 w=100;
CHANNEL cd from b1 4 to m0 4 w=100;
CHANNEL ce from b1 5 to m0 5 w=100;
CHANNEL cf from b1 6 to m0 6 w=100;
CHANNEL cg from b1 7 to m0 7 w=100;
CHANNEL ch from b1 8 to m0 8 w=100;

CHANNEL c1 from m0 9 to m1 1 w=100;
CHANNEL c2 from m1 2 to ct1 1 w=100;
CHANNEL c3 from m1 3 to ct2 1 w=100;
CHANNEL c4 from m1 4 to ct3 1 w=100;
CHANNEL c5 from m1 5 to ct4 1 w=100;
CHANNEL c6 from m1 6 to ct5 1 w=100;
CHANNEL c7 from m1 7 to ct6 1 w=100;
CHANNEL c8 from m1 8 to ct7 1 w=100;
CHANNEL c9 from m1 9 to ct8 1 w=100;

CHANNEL c10 from ct1 2 to ct2 4 w=100;
CHANNEL c11 from ct2 2 to ct3 4 w=100;
CHANNEL c12 from ct3 2 to ct4 4 w=100;
CHANNEL c13 from ct4 2 to ct5 4 w=100;
```

```
CHANNEL c14 from ct5 2 to ct6 4 w=100;
CHANNEL c15 from ct6 2 to ct7 4 w=100;
CHANNEL c16 from ct7 2 to ct8 4 w=100;

CHANNEL c17 from ct1 3 to ct9 1 w=100;
CHANNEL c18 from ct2 3 to ct10 1 w=100;
CHANNEL c19 from ct3 3 to ct11 1 w=100;
CHANNEL c20 from ct4 3 to ct12 1 w=100;
CHANNEL c21 from ct5 3 to ct13 1 w=100;
CHANNEL c22 from ct6 3 to ct14 1 w=100;
CHANNEL c23 from ct7 3 to ct15 1 w=100;
CHANNEL c24 from ct8 3 to ct16 1 w=100;

CHANNEL c25 from ct9 2 to ct10 4 w=100;
CHANNEL c26 from ct10 2 to ct11 4 w=100;
CHANNEL c27 from ct11 2 to ct12 4 w=100;
CHANNEL c28 from ct12 2 to ct13 4 w=100;
CHANNEL c29 from ct13 2 to ct14 4 w=100;
CHANNEL c30 from ct14 2 to ct15 4 w=100;
CHANNEL c31 from ct15 2 to ct16 4 w=100;

CHANNEL c32 from ct9 3 to ct17 1 w=100;
CHANNEL c33 from ct10 3 to ct18 1 w=100;
CHANNEL c34 from ct11 3 to ct19 1 w=100;
CHANNEL c35 from ct12 3 to ct20 1 w=100;
CHANNEL c36 from ct13 3 to ct21 1 w=100;
CHANNEL c37 from ct14 3 to ct22 1 w=100;
CHANNEL c38 from ct15 3 to ct23 1 w=100;
CHANNEL c39 from ct16 3 to ct24 1 w=100;

CHANNEL c40 from ct17 2 to ct18 4 w=100;
CHANNEL c41 from ct18 2 to ct19 4 w=100;
CHANNEL c42 from ct19 2 to ct20 4 w=100;
CHANNEL c43 from ct20 2 to ct21 4 w=100;
CHANNEL c44 from ct21 2 to ct22 4 w=100;
CHANNEL c45 from ct22 2 to ct23 4 w=100;
CHANNEL c46 from ct23 2 to ct24 4 w=100;
```

```
CHANNEL c47 from ct17 3 to ct25 1 w=100;
CHANNEL c48 from ct18 3 to ct26 1 w=100;
CHANNEL c49 from ct19 3 to ct27 1 w=100;
CHANNEL c50 from ct20 3 to ct28 1 w=100;
CHANNEL c51 from ct21 3 to ct29 1 w=100;
CHANNEL c52 from ct22 3 to ct30 1 w=100;
CHANNEL c53 from ct23 3 to ct31 1 w=100;
CHANNEL c54 from ct24 3 to ct32 1 w=100;

CHANNEL c55 from ct25 2 to ct26 4 w=100;
CHANNEL c56 from ct26 2 to ct27 4 w=100;
CHANNEL c57 from ct27 2 to ct28 4 w=100;
CHANNEL c58 from ct28 2 to ct29 4 w=100;
CHANNEL c59 from ct29 2 to ct30 4 w=100;
CHANNEL c60 from ct30 2 to ct31 4 w=100;
CHANNEL c61 from ct31 2 to ct32 4 w=100;

CHANNEL c62 from ct25 3 to ct33 1 w=100;
CHANNEL c63 from ct26 3 to ct34 1 w=100;
CHANNEL c64 from ct27 3 to ct35 1 w=100;
CHANNEL c65 from ct28 3 to ct36 1 w=100;
CHANNEL c66 from ct29 3 to ct37 1 w=100;
CHANNEL c67 from ct30 3 to ct38 1 w=100;
CHANNEL c68 from ct31 3 to ct39 1 w=100;
CHANNEL c69 from ct32 3 to ct40 1 w=100;

CHANNEL c70 from ct33 2 to ct34 4 w=100;
CHANNEL c71 from ct34 2 to ct35 4 w=100;
CHANNEL c72 from ct35 2 to ct36 4 w=100;
CHANNEL c73 from ct36 2 to ct37 4 w=100;
CHANNEL c74 from ct37 2 to ct38 4 w=100;
CHANNEL c75 from ct38 2 to ct39 4 w=100;
CHANNEL c76 from ct39 2 to ct40 4 w=100;

CHANNEL c77 from ct33 3 to ct41 1 w=100;
CHANNEL c78 from ct34 3 to ct42 1 w=100;
```

```
CHANNEL c79 from ct35 3 to ct43 1 w=100;
CHANNEL c80 from ct36 3 to ct44 1 w=100;
CHANNEL c81 from ct37 3 to ct45 1 w=100;
CHANNEL c82 from ct38 3 to ct46 1 w=100;
CHANNEL c83 from ct39 3 to ct47 1 w=100;
CHANNEL c84 from ct40 3 to ct48 1 w=100;

CHANNEL c85 from ct41 2 to ct42 4 w=100;
CHANNEL c86 from ct42 2 to ct43 4 w=100;
CHANNEL c87 from ct43 2 to ct44 4 w=100;
CHANNEL c88 from ct44 2 to ct45 4 w=100;
CHANNEL c89 from ct45 2 to ct46 4 w=100;
CHANNEL c90 from ct46 2 to ct47 4 w=100;
CHANNEL c91 from ct47 2 to ct48 4 w=100;

CHANNEL c92 from ct41 3 to ct49 1 w=100;
CHANNEL c93 from ct42 3 to ct50 1 w=100;
CHANNEL c94 from ct43 3 to ct51 1 w=100;
CHANNEL c95 from ct44 3 to ct52 1 w=100;
CHANNEL c96 from ct45 3 to ct53 1 w=100;
CHANNEL c97 from ct46 3 to ct54 1 w=100;
CHANNEL c98 from ct47 3 to ct55 1 w=100;
CHANNEL c99 from ct48 3 to ct56 1 w=100;

CHANNEL c100 from ct49 2 to ct50 4 w=100;
CHANNEL c101 from ct50 2 to ct51 4 w=100;
CHANNEL c102 from ct51 2 to ct52 4 w=100;
CHANNEL c103 from ct52 2 to ct53 4 w=100;
CHANNEL c104 from ct53 2 to ct54 4 w=100;
CHANNEL c105 from ct54 2 to ct55 4 w=100;
CHANNEL c106 from ct55 2 to ct56 4 w=100;

CHANNEL c107 from ct49 3 to ct57 1 w=100;
CHANNEL c108 from ct50 3 to ct58 1 w=100;
CHANNEL c109 from ct51 3 to ct59 1 w=100;
CHANNEL c110 from ct52 3 to ct60 1 w=100;
CHANNEL c111 from ct53 3 to ct61 1 w=100;
```

```
CHANNEL c112 from ct54 3 to ct62 1 w=100;
CHANNEL c113 from ct55 3 to ct63 1 w=100;
CHANNEL c114 from ct56 3 to ct64 1 w=100;

CHANNEL c115 from ct57 2 to ct58 4 w=100;
CHANNEL c116 from ct58 2 to ct59 4 w=100;
CHANNEL c117 from ct59 2 to ct60 4 w=100;
CHANNEL c118 from ct60 2 to ct61 4 w=100;
CHANNEL c119 from ct61 2 to ct62 4 w=100;
CHANNEL c120 from ct62 2 to ct63 4 w=100;
CHANNEL c121 from ct63 2 to ct64 4 w=100;
CHANNEL c122 from ct57 3 to m2 1 w=100;
CHANNEL c123 from ct58 3 to m2 2 w=100;
CHANNEL c124 from ct59 3 to m2 3 w=100;
CHANNEL c125 from ct60 3 to m2 4 w=100;
CHANNEL c126 from ct61 3 to m2 5 w=100;
CHANNEL c127 from ct62 3 to m2 6 w=100;
CHANNEL c128 from ct63 3 to m2 7 w=100;
CHANNEL c129 from ct64 3 to m2 8 w=100;
CHANNEL c130 from m2 9 to p2 1 w=100;

END LAYER

LAYER CONTROL
V BANK cpb1 of 8 PORT r=100 dir=RIGHT spacing=1500 channelWidth=50;
V BANK cpb2 of 7 PORT r=100 dir=LEFT spacing=1500 channelWidth=50;

VALVE v1 on c10 w=100 l=300;
VALVE v2 on c11 w=100 l=300;
VALVE v3 on c12 w=100 l=300;
VALVE v4 on c13 w=100 l=300;
VALVE v5 on c14 w=100 l=300;
VALVE v6 on c15 w=100 l=300;
VALVE v7 on c16 w=100 l=300;

NET n1 from cpb1 1 to v1 3, v2 3, v3 3, v4 3, v5 3, v6 3, v7 3 channelWidth
   =50;
```

```
VALVE v8 on c17 w=300 l=100;
VALVE v9 on c18 w=300 l=100;
VALVE v10 on c19 w=300 l=100;
VALVE v11 on c20 w=300 l=100;
VALVE v12 on c21 w=300 l=100;
VALVE v13 on c22 w=300 l=100;
VALVE v14 on c23 w=300 l=100;
VALVE v15 on c24 w=300 l=100;

CHANNEL cc1 from v8 2 to v9 4 w=50;
CHANNEL cc2 from v9 2 to v10 4 w=50;
CHANNEL cc3 from v10 2 to v11 4 w=50;
CHANNEL cc4 from v11 2 to v12 4 w=50;
CHANNEL cc5 from v12 2 to v13 4 w=50;
CHANNEL cc6 from v13 2 to v14 4 w=50;
CHANNEL cc7 from v14 2 to v15 4 w=50;
CHANNEL cc8 from v15 2 to cpb2 1 w=50;

VALVE v16 on c25 w=100 l=300;
VALVE v17 on c26 w=100 l=300;
VALVE v18 on c27 w=100 l=300;
VALVE v19 on c28 w=100 l=300;
VALVE v20 on c29 w=100 l=300;
VALVE v21 on c30 w=100 l=300;
VALVE v22 on c31 w=100 l=300;

NET n2 from cpb1 2 to v16 3, v17 3, v18 3, v19 3, v20 3, v21 3, v22 3
    channelWidth=50;

VALVE v23 on c32 w=300 l=100;
VALVE v24 on c33 w=300 l=100;
VALVE v25 on c34 w=300 l=100;
VALVE v26 on c35 w=300 l=100;
VALVE v27 on c36 w=300 l=100;
VALVE v28 on c37 w=300 l=100;
VALVE v29 on c38 w=300 l=100;
```

```
VALVE v30 on c39 w=300 l=100;

CHANNEL cc9 from v23 2 to v24 4 w=50;
CHANNEL cc10 from v24 2 to v25 4 w=50;
CHANNEL cc11 from v25 2 to v26 4 w=50;
CHANNEL cc12 from v26 2 to v27 4 w=50;
CHANNEL cc13 from v27 2 to v28 4 w=50;
CHANNEL cc14 from v28 2 to v29 4 w=50;
CHANNEL cc15 from v29 2 to v30 4 w=50;
CHANNEL cc16 from v30 2 to cpb2 2 w=50;

VALVE v31 on c40 w=100 l=300;
VALVE v32 on c41 w=100 l=300;
VALVE v33 on c42 w=100 l=300;
VALVE v34 on c43 w=100 l=300;
VALVE v35 on c44 w=100 l=300;
VALVE v36 on c45 w=100 l=300;
VALVE v37 on c46 w=100 l=300;

NET n3 from cpb1 3 to v31 3, v32 3, v33 3, v34 3, v35 3, v36 3, v37 3
    channelWidth=50;

VALVE v38 on c47 w=300 l=100;
VALVE v39 on c48 w=300 l=100;
VALVE v40 on c49 w=300 l=100;
VALVE v41 on c50 w=300 l=100;
VALVE v42 on c51 w=300 l=100;
VALVE v43 on c52 w=300 l=100;
VALVE v44 on c53 w=300 l=100;
VALVE v45 on c54 w=300 l=100;

CHANNEL cc17 from v38 2 to v39 4 w=50;
CHANNEL cc18 from v39 2 to v40 4 w=50;
CHANNEL cc19 from v40 2 to v41 4 w=50;
CHANNEL cc20 from v41 2 to v42 4 w=50;
CHANNEL cc21 from v42 2 to v43 4 w=50;
CHANNEL cc22 from v43 2 to v44 4 w=50;
```

```
CHANNEL cc23 from v44 2 to v45 4 w=50;
CHANNEL cc24 from v45 2 to cpb2 3 w=50;

VALVE v46 on c55 w=100 l=300;
VALVE v47 on c56 w=100 l=300;
VALVE v48 on c57 w=100 l=300;
VALVE v49 on c58 w=100 l=300;
VALVE v50 on c59 w=100 l=300;
VALVE v51 on c60 w=100 l=300;
VALVE v52 on c61 w=100 l=300;

NET n4 from cpb1 4 to v46 3, v47 3, v48 3, v49 3, v50 3, v51 3, v52 3
    channelWidth=50;

VALVE v53 on c62 w=300 l=100;
VALVE v54 on c63 w=300 l=100;
VALVE v55 on c64 w=300 l=100;
VALVE v56 on c65 w=300 l=100;
VALVE v57 on c66 w=300 l=100;
VALVE v58 on c67 w=300 l=100;
VALVE v59 on c68 w=300 l=100;
VALVE v60 on c69 w=300 l=100;

CHANNEL cc25 from v53 2 to v54 4 w=50;
CHANNEL cc26 from v54 2 to v55 4 w=50;
CHANNEL cc27 from v55 2 to v56 4 w=50;
CHANNEL cc28 from v56 2 to v57 4 w=50;
CHANNEL cc29 from v57 2 to v58 4 w=50;
CHANNEL cc30 from v58 2 to v59 4 w=50;
CHANNEL cc31 from v59 2 to v60 4 w=50;
CHANNEL cc32 from v60 2 to cpb2 4 w=50;

VALVE v61 on c70 w=100 l=300;
VALVE v62 on c71 w=100 l=300;
VALVE v63 on c72 w=100 l=300;
VALVE v64 on c73 w=100 l=300;
VALVE v65 on c74 w=100 l=300;
```

```
VALVE v66 on c75 w=100 l=300;
VALVE v67 on c76 w=100 l=300;

NET n5 from cpb1 5 to v61 3, v62 3, v63 3, v64 3, v65 3, v66 3, v67 3
    channelWidth=50;

VALVE v68 on c77 w=300 l=100;
VALVE v69 on c78 w=300 l=100;
VALVE v70 on c79 w=300 l=100;
VALVE v71 on c80 w=300 l=100;
VALVE v72 on c81 w=300 l=100;
VALVE v73 on c82 w=300 l=100;
VALVE v74 on c83 w=300 l=100;
VALVE v75 on c84 w=300 l=100;

CHANNEL cc33 from v68 2 to v69 4 w=50;
CHANNEL cc34 from v69 2 to v70 4 w=50;
CHANNEL cc35 from v70 2 to v71 4 w=50;
CHANNEL cc36 from v71 2 to v72 4 w=50;
CHANNEL cc37 from v72 2 to v73 4 w=50;
CHANNEL cc38 from v73 2 to v74 4 w=50;
CHANNEL cc39 from v74 2 to v75 4 w=50;
CHANNEL cc40 from v75 2 to cpb2 5 w=50;

VALVE v76 on c85 w=100 l=300;
VALVE v77 on c86 w=100 l=300;
VALVE v78 on c87 w=100 l=300;
VALVE v79 on c88 w=100 l=300;
VALVE v80 on c89 w=100 l=300;
VALVE v81 on c90 w=100 l=300;
VALVE v82 on c91 w=100 l=300;

NET n6 from cpb1 6 to v76 3, v77 3, v78 3, v79 3, v80 3, v81 3, v82 3
    channelWidth=50;

VALVE v83 on c92 w=300 l=100;
VALVE v84 on c93 w=300 l=100;
```

```
VALVE v85 on c94 w=300 l=100;
VALVE v86 on c95 w=300 l=100;
VALVE v87 on c96 w=300 l=100;
VALVE v88 on c97 w=300 l=100;
VALVE v89 on c98 w=300 l=100;
VALVE v90 on c99 w=300 l=100;

CHANNEL cc41 from v83 2 to v84 4 w=50;
CHANNEL cc42 from v84 2 to v85 4 w=50;
CHANNEL cc43 from v85 2 to v86 4 w=50;
CHANNEL cc44 from v86 2 to v87 4 w=50;
CHANNEL cc45 from v87 2 to v88 4 w=50;
CHANNEL cc46 from v88 2 to v89 4 w=50;
CHANNEL cc47 from v89 2 to v90 4 w=50;
CHANNEL cc48 from v90 2 to cpb2 6 w=50;

VALVE v91 on c100 w=100 l=300;
VALVE v92 on c101 w=100 l=300;
VALVE v93 on c102 w=100 l=300;
VALVE v94 on c103 w=100 l=300;
VALVE v95 on c104 w=100 l=300;
VALVE v96 on c105 w=100 l=300;
VALVE v97 on c106 w=100 l=300;

NET n7 from cpb1 7 to v91 3, v92 3, v93 3, v94 3, v95 3, v96 3, v97 3
    channelWidth=50;

VALVE v98 on c107 w=300 l=100;
VALVE v99 on c108 w=300 l=100;
VALVE v100 on c109 w=300 l=100;
VALVE v101 on c110 w=300 l=100;
VALVE v102 on c111 w=300 l=100;
VALVE v103 on c112 w=300 l=100;
VALVE v104 on c113 w=300 l=100;
VALVE v105 on c114 w=300 l=100;

CHANNEL cc49 from v98 2 to v99 4 w=50;
```

```
CHANNEL cc50 from v99 2 to v100 4 w=50;
CHANNEL cc51 from v100 2 to v101 4 w=50;
CHANNEL cc52 from v101 2 to v102 4 w=50;
CHANNEL cc53 from v102 2 to v103 4 w=50;
CHANNEL cc54 from v103 2 to v104 4 w=50;
CHANNEL cc55 from v104 2 to v105 4 w=50;
CHANNEL cc56 from v105 2 to cpb2 7 w=50;

VALVE v106 on c115 w=100 l=300;
VALVE v107 on c116 w=100 l=300;
VALVE v108 on c117 w=100 l=300;
VALVE v109 on c118 w=100 l=300;
VALVE v110 on c119 w=100 l=300;
VALVE v111 on c120 w=100 l=300;
VALVE v112 on c121 w=100 l=300;

NET n8 from cpb1 8 to v106 3, v107 3, v108 3, v109 3, v110 3, v111 3, v112
    3 channelWidth=50;

V BANK cb3 of 3 PORT r=100 dir=RIGHT spacing=1200 channelWidth=50;
V BANK cb4 of 3 PORT r=100 dir=LEFT spacing=1200 channelWidth=50;
CHANNEL cc57 from m0 10 to cb3 1 w=50;
CHANNEL cc58 from m0 12 to cb3 2 w=50;
CHANNEL cc59 from m0 14 to cb3 3 w=50;
CHANNEL cc60 from m0 11 to cb4 1 w=50;
CHANNEL cc61 from m0 13 to cb4 2 w=50;
CHANNEL cc62 from m0 15 to cb4 3 w=50;
END LAYER
```

B.12 Device txtl

```
3D DEVICE txtl

LAYER FLOW
PORT p1 r=400;
PORT p2 r=400;
PORT p3 r=400;
NODE n1;
```

```
V MIXER m1 numBends=10 bendSpacing=500 bendLength=20000 channelWidth=400;
H 3DVALVE v1 radius=500 gap=600;
H 3DVALVE v2 radius=500 gap=600;

CHANNEL c1 from p1 2 to v1 4 w=400;
CHANNEL c2 from v1 2 to n1 4 w=400;
CHANNEL c3 from p2 4 to v2 2 w=400;
CHANNEL c4 from v2 4 to n1 2 w=400;
CHANNEL c5 from n1 3 to m1 1 w=400;
CHANNEL c6 from m1 2 to p3 1 w=400;

END LAYER
```

B.13 Device transposer-cells

```
3D DEVICE transposer_cells

LAYER FLOW
TRANSPOSER r1 valveRadius=1000 valveGap=400 flowChannelWidth=800
    controlChannelWidth=800;
PORT p1, p2 r=800;

V BANK b1 of 4 PORT r=800 dir=RIGHT spacing=9000 channelWidth=800;

V TREE t1 2 to 1 spacing=9000 flowChannelWidth=800;
V TREE t2 2 to 1 spacing=9000 flowChannelWidth=800;

H MIXER x1 numBends=5 bendSpacing=400 bendLength=10000 channelWidth=800;
H MIXER x2 numBends=5 bendSpacing=400 bendLength=10000 channelWidth=800;

CHANNEL c1 from b1 1 to t1 1 w=800;
CHANNEL c2 from b1 2 to t1 2 w=800;
CHANNEL c3 from b1 3 to t2 1 w=800;
CHANNEL c4 from b1 4 to t2 2 w=800;
CHANNEL c5 from t1 3 to x1 1 w=800;
CHANNEL c6 from t2 3 to x2 1 w=800;
CHANNEL c7 from x1 2 to r1 1 w=800;
CHANNEL c8 from x2 2 to r1 2 w=800;
```

```
CHANNEL c9 from r1 3 to p1 4 w=800;
CHANNEL c10 from r1 4 to p2 4 w=800;

END LAYER
```

B.14 Device transposer-web

```
3D DEVICE transposer_web

LAYER FLOW
TRANSPOSER t1 valveRadius=1000 valveGap=400 flowChannelWidth=800
    controlChannelWidth=800;
TRANSPOSER t2 valveRadius=1000 valveGap=400 flowChannelWidth=800
    controlChannelWidth=800;
TRANSPOSER t3 valveRadius=1000 valveGap=400 flowChannelWidth=800
    controlChannelWidth=800;
TRANSPOSER t4 valveRadius=1000 valveGap=400 flowChannelWidth=800
    controlChannelWidth=800;
TRANSPOSER t5 valveRadius=1000 valveGap=400 flowChannelWidth=800
    controlChannelWidth=800;

CHANNEL c1 from t1 3 to t2 1 w=800;
CHANNEL c2 from t1 4 to t3 1 w=800;
CHANNEL c3 from t3 3 to t2 2 w=800;
CHANNEL c4 from t4 3 to t3 2 w=800;
CHANNEL c5 from t3 4 to t5 1 w=800;
CHANNEL c6 from t4 4 to t5 2 w=800;
END LAYER
```

B.15 Device mixer-3d

```
3D DEVICE mixer_3d

LAYER FLOW
H BANK pb1 of 3 PORT r=600 dir=DOWN spacing=10000 channelWidth=800;
NODE n1, n2, n3;
PORT p1 r=600;
V GRADIENT GENERATOR g1 2 to 6 numBends=5 bendSpacing=500 bendLength=2000
```

```
   channelWidth=800;
V 3DVALVE v1 radius=1000 gap=400;
V 3DVALVE v2 radius=1000 gap=400;
V 3DVALVE v3 radius=1000 gap=400;

CHANNEL c1 from pb1 1 to v1 1 w=800;
CHANNEL c2 from pb1 2 to v2 1 w=800;
CHANNEL c3 from pb1 3 to v3 1 w=800;
CHANNEL c4 from v1 3 to n1 4 w=800;
CHANNEL c5 from v2 3 to n2 1 w=800;
CHANNEL c6 from v3 3 to n3 2 w=800;
CHANNEL c7 from n1 2 to n2 4 w=800;
CHANNEL c8 from n2 2 to n3 4 w=800;
CHANNEL c9 from n1 3 to g1 1 w=800;
CHANNEL c10 from n3 3 to g1 2 w=800;
CHANNEL c11 from g1 3 to p1 1 w=2400;
END LAYER
```

Appendix C

Photolithography procedure

C.1 Setup

1. Set hot plates to 75C and 103C.
2. Wash wafers with the following. Hold wafer with flat tweezers, never with hands.
   (a) Acetone
   (b) Isopropanol
   (c) Ethyl Alcohol
   (d) Acetone
   (e) DI water
3. Air gun to dry.
4. Place in oven at 120C. Hard bake 10-15 minutes to fully dry.
5. Cut mask transparencies down to size, and tape transparencies to glass slides.
6. Oxygen plasma treatment: power=150, flow=300, time=3 minutes.

C.2 Control layer

1. Load mask into mask aligner with alignment gap of 30 μm.

2. Spin SU-8 10 at 1000RPM for 60 seconds.

3. Bake 2.5 minutes at 65C followed by 6 minutes at 95C.

4. Expose hard contact for 8 seconds.

5. Post exposure bake for 1 minute 65C, 2.5 minutes at 95C.

6. Develop for 2-4 minutes Check by eye to keep from overdeveloping.

7. Hard bake for 15-20 minutes at 120-180C.

C.3 Flow layer

1. Load mask for cell traps into mask aligner with alignment gap of 30 μm.

2. Spin coat with SU-8 2. The height of the cell traps depends on the spin speed.

   (a) 750RPM for 60 seconds: h=4.2 μm

   (b) 820RPM for 60 seconds: h=3.8 μm

   (c) 900RPM for 60 seconds: h=3.5 μm

   (d) 1800RPM for 60 seconds: h=3 μm

3. Bake for 1 minute at 65C, then 3 minutes at 95C.

4. Expose hard contact for 8 seconds.

5. Post exposure bake for 1 minute at 65C then 1 minute at 95C.

6. Develop for 1 minute or more.

7. Spin coat with HDMS (surface adhesion promoter) at 1000RPM for 60 seconds.

8. Bake at 95C for 12 minutes.

9. Spin coat A24629 (photoresist) at 3400RPM for 60 seconds.

10. Bake at 90C for 10 minutes.

11. Expose hard contact for 20 seconds.

12. Develop for 8 minutes or more. Use diluted 1:4 developer.

13. Bake at 150C for 1 minute to reflow channels.

Appendix D

PDMS molding

D.1 Control layer

1. Mix 60g gel and 6g curing agent for at least 3 minutes.
2. Pour in plate over wafer.
3. Place cap lightly over plate, leaving a small gap.
4. Place in vacuum for 5 minutes before releasing vacuum. Wait until all bubbles have dispersed. Repeat as needed.
5. Place in oven at 80C for 3-4 hours.
6. Cut along guide lines, and punch holes.

D.2 Flow layer

1. Place wafer in a glass plate on top of supports.
2. Apply 50 μL of silane to wafer by placing it at the edge of the glass plate.
3. Close lid of plate and wait 10 minutes for silane to coat wafer.
4. Open lid and wait 3 minutes.
5. Mix 20g gel with 2g curing agent for at least 3 minutes and degas.
6. Place wafer in spin coater.

7. Pour PDMS over wafer, and spin at 3500RPM for 1 minute to coat.
8. Place in oven at 80C for 10-12 minutes.
9. Pull wafer out of the oven while the PDMS is still sticky.
10. Align cut control layer with flow layer by eye under microscope.
11. Place in oven for 3-4 hours.
12. Cut along guide lines, and punch holes in flow layer.

D.3 Bonding to glass

1. Clean device with scotch tape.
2. Wash glass cover slide.
3. Open the compressed O2 valve on your tank and make sure the flow through the UVO cleaner is 0.4 - 0.6 scfm.
4. Warm up the UVO cleaner, by running it for 5 minutes.
5. Once the warm up is done, open the loading tray, there should be a faint smell of ozone.
6. Place the chips with feature side up and coverslips onto the tray.
7. Close the tray and run the bonder for 3 minutes. The chamber should appear magenta through the holes.
8. Carefully place device feature side down on glass and remove all air bubbles.
9. Place in oven at 80C overnight.

References

Amin, N., Thies, W., and Amarasinghe, S. (2009). Computer-aided design for microfluidic chips based on multilayer soft lithography. In *IEEE International Conference on Computer Design, 2009. ICCD 2009.*, pages 2–9. IEEE.

Anderson, J., Dueber, J. E., Leguia, M., Wu, G. C., Goler, J. A., Arkin, A. P., and Keasling, J. D. (2010). Bglbricks: A flexible standard for biological part assembly. *Journal of biological engineering*, 4(1):1–12.

Anderson, J. C., Clarke, E. J., Arkin, A. P., and Voigt, C. A. (2006). Environmentally controlled invasion of cancer cells by engineered bacteria. *Journal of molecular biology*, 355(4):619–627.

Andrianantoandro, E., Basu, S., Karig, D. K., and Weiss, R. (2006). Synthetic biology: new engineering rules for an emerging discipline. *Molecular systems biology*, 2(1):2006.0028.

Appleton, E., Tao, J., Haddock, T., and Densmore, D. (2014). Interactive assembly algorithms for molecular cloning. *Nature methods*, 11:657–662.

Araci, I. E. and Brisk, P. (2014). Recent developments in microfluidic large scale integration. *Current Opinion in Biotechnology*, 25:60–68.

Balaban, N. Q., Merrin, J., Chait, R., Kowalik, L., and Leibler, S. (2004). Bacterial persistence as a phenotypic switch. *Science*, 305(5690):1622–1625.

Balagaddé, F. K., You, L., Hansen, C. L., Arnold, F. H., and Quake, S. R. (2005). Long-term monitoring of bacteria undergoing programmed population control in a microchemostat. *Science*, 309(5731):137–140.

Basu, S., Gerchman, Y., Collins, C. H., Arnold, F. H., and Weiss, R. (2005). A synthetic multicellular system for programmed pattern formation. *Nature*, 434(7037):1130–1134.

Baumgardner, J., Acker, K., Adefuye, O., Crowley, S. T., DeLoache, W., Dickson, J. O., Heard, L., Martens, A. T., Morton, N., Ritter, M., et al. (2009). Solving a hamiltonian path problem with a bacterial computer. *Journal of biological engineering*, 3(11):1–11.

Baumgartner, B. L., Bennett, M. R., Ferry, M., Johnson, T. L., Tsimring, L. S., and Hasty, J. (2011). Antagonistic gene transcripts regulate adaptation to new growth environments. *Proceedings of the National Academy of Sciences*, 108(52):21087–21092.

Beal, J., Lu, T., and Weiss, R. (2011). Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks. *PLoS One*, 6(8):e22490.

Beal, J., Weiss, R., Densmore, D., Adler, A., Appleton, E., Babb, J., Bhatia, S., Davidsohn, N., Haddock, T., Loyall, J., et al. (2012). An end-to-end workflow for engineering of biological networks from high-level specifications. *ACS synthetic biology*, 1(8):317–331.

Becker, H. and Gärtner, C. (2008). Polymer microfabrication technologies for microfluidic systems. *Analytical and bioanalytical chemistry*, 390(1):89–111.

Bennett, M. R. and Hasty, J. (2009). Microfluidic devices for measuring gene network dynamics in single cells. *Nature Reviews Genetics*, 10(9):628–638.

Bennett, M. R., Pang, W. L., Ostroff, N. A., Baumgartner, B. L., Nayak, S., Tsimring, L. S., and Hasty, J. (2008). Metabolic gene regulation in a dynamically changing environment. *Nature*, 454(7208):1119–1122.

Berg, M. d., Cheong, O., Kreveld, M. v., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition.

Betz, V. and Rose, J. (1997). VPR: A new packing, placement, and routing tool for fpga research. In *International Workshop on Field Programmable Logic and Appliation*.

Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011a). Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS one*, 6(4):e18882.

Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011b). Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS one*, 6(4):e18882.

Bonnet, J., Subsoontorn, P., and Endy, D. (2012). Rewritable digital data storage in live cells via engineered control of recombination directionality. *Proceedings of the National Academy of Sciences*, 109(23):8884–8889.

Bonnet, J., Yin, P., Ortiz, M. E., Subsoontorn, P., and Endy, D. (2013). Amplifying genetic logic gates. *Science*, 340(6132):599–603.

Breslauer, D. N., Lee, P. J., and Lee, L. P. (2006). Microfluidics-based systems biology. *Molecular Biosystems*, 2(2):97–112.

Brophy, J. A. and Voigt, C. A. (2014). Principles of genetic circuit design. *Nature methods*, 11(5):508–520.

Cai, Y., Wilson, M. L., and Peccoud, J. (2010). Genocad for igem: a grammatical approach to the design of standard-compliant constructs. *Nucleic acids research*, 38(8):2637–2644.

Canton, B., Labno, A., and Endy, D. (2008). Refinement and standardization of synthetic biological parts and devices. *Nature biotechnology*, 26(7):787–793.

Carr, P. A. and Church, G. M. (2009). Genome engineering. *Nature biotechnology*, 27(12):1151–1162.

Casini, A., MacDonald, J. T., De Jonghe, J., Christodoulou, G., Freemont, P. S., Baldwin, G. S., and Ellis, T. (2014). One-pot dna construction for synthetic biology: the modular overlap-directed assembly with linkers (modal) strategy. *Nucleic acids research*, 42(1):e7–e7.

Chakrabarty, K. (2010). Design automation and test solutions for digital microfluidic biochips. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 57(1):4–17.

Chakrabarty, K. and Zeng, J. (2005). Design automation for microfluidics-based biochips. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 1(3):186–223.

Chandran, D., Bergmann, F. T., Sauro, H. M., et al. (2009). Tinkercell: modular cad tool for synthetic biology. *Journal of biological engineering*, 3(1):19.

Chang, C., Bharadwaj, R., Singh, A., Chandrasekaran, A., and Hillson, N. (2012). Microfluidic platform for synthetic biology applications. US Patent App. 13/437,727.

Chen, Y.-J., Liu, P., Nielsen, A. A., Brophy, J. A., Clancy, K., Peterson, T., and Voigt, C. A. (2013). Characterization of 582 natural and synthetic terminators and quantification of their design constraints. *Nature methods*, 10(7):659–664.

Cheng, A. A. and Lu, T. K. (2012). Synthetic biology: an emerging engineering discipline. *Annual review of biomedical engineering*, 14:155–178.

Cheong, R., Wang, C. J., and Levchenko, A. (2009a). High content cell screening in a microfluidic device. *Molecular & Cellular Proteomics*, 8(3):433–442.

Cheong, R., Wang, C. J., and Levchenko, A. (2009b). Using a microfluidic device for high-content analysis of cell signaling. *Science signaling*, 2(75):pl2.

Chou, H.-P., Unger, M. A., and Quake, S. R. (2001). A microfabricated rotary pump. *Biomedical Microdevices*, 3(4):323–330.

Cobb, R. E., Si, T., and Zhao, H. (2012). Directed evolution: an evolving and enabling synthetic biology tool. *Current opinion in chemical biology*, 16(3):285–291.

Collins, C. H., Leadbetter, J. R., and Arnold, F. H. (2006). Dual selection enhances the signaling specificity of a variant of the quorum-sensing transcriptional activator luxr. *Nature biotechnology*, 24(6):708–712.

Cooksey, G. A., Sip, C. G., and Folch, A. (2009). A multi-purpose microfluidic perfusion system with combinatorial choice of inputs, mixtures, gradient patterns, and flow rates. *Lab on a Chip*, 9(3):417–426.

Cox, C. D., McCollum, J. M., Austin, D. W., Allen, M. S., Dar, R. D., and Simpson, M. L. (2006). Frequency domain analysis of noise in simple gene circuits. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 16(2):026102.

Daniel, R., Rubens, J. R., Sarpeshkar, R., and Lu, T. K. (2013). Synthetic analog computation in living cells. *Nature*, 497:619–623.

Danino, T., Mondragón-Palomino, O., Tsimring, L., and Hasty, J. (2010). A synchronized quorum of genetic clocks. *Nature*, 463(7279):326–330.

Dénervaud, N., Becker, J., Delgado-Gonzalo, R., Damay, P., Rajkumar, A. S., Unser, M., Shore, D., Naef, F., and Maerkl, S. J. (2013). A chemostat array enables the spatio-temporal analysis of the yeast proteome. *Proceedings of the National Academy of Sciences*, 110(39):15842–15847.

Densmore, D. and Hassoun, S. (2012). Design automation for synthetic biological systems. *IEEE Design and Test of Computers*, 29(3):7–20.

Densmore, D., Hsiau, T. H.-C., Kittleson, J. T., DeLoache, W., Batten, C., and Anderson, J. C. (2010). Algorithms for automated dna assembly. *Nucleic acids research*, 38(8):2607–2616.

Dertinger, S. K., Chiu, D. T., Jeon, N. L., and Whitesides, G. M. (2001). Generation of gradients having complex shapes using microfluidic networks. *Analytical Chemistry*, 73(6):1240–1246.

Devaraju, N. S. G. K. and Unger, M. A. (2012). Pressure driven digital logic in pdms based microfluidic devices fabricated by multilayer soft lithography. *Lab on a Chip*, 12(22):4809–4815.

Duffy, D. C., McDonald, J. C., Schueller, O. J., and Whitesides, G. M. (1998). Rapid prototyping of microfluidic systems in poly (dimethylsiloxane). *Analytical chemistry*, 70(23):4974–4984.

Duncan, P. N., Ahrar, S., and Hui, E. E. (2015). Scaling of pneumatic digital logic circuits. *Lab on a Chip*, 15(5):1360–1365.

Duncan, P. N., Nguyen, T. V., and Hui, E. E. (2013). Pneumatic oscillator circuits for timing and control of integrated microfluidics. *Proceedings of the National Academy of Sciences*, 110(45):18104–18109.

Ebeling, C., McMurchie, L., Hauck, S. A., and Burns, S. (1995). Placement and routing tools for the triptych FPGA. *IEEE Transactions on VLSI Systems*, pages 473 – 482.

Ellis, T., Adie, T., and Baldwin, G. S. (2011). Dna assembly for synthetic biology: from parts to pathways and beyond. *Integrative Biology*, 3(2):109–118.

Elowitz, M. B. and Leibler, S. (2000). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338.

Elveflow (2015). Elveflow. `http://www.elveflow.com/`. Accessed: 2014-06-17.

Endy, D. (2005). Foundations for engineering biology. *Nature*, 438(7067):449–453.

Engler, C., Gruetzner, R., Kandzia, R., and Marillonnet, S. (2009). Golden gate shuffling: a one-pot dna shuffling method based on type iis restriction enzymes. *PLoS one*, 4(5):e5553.

Engler, C., Kandzia, R., and Marillonnet, S. (2008). A one pot, one step, precision cloning method with high throughput capability. *PLoS one*, 3(11):e3647.

Ferry, M., Razinkov, I., and Hasty, J. (2011). Microfluidics for synthetic biology from design to execution. *Methods in enzymology*, 497:295.

Fidalgo, L. M. and Maerkl, S. J. (2011). A software-programmable microfluidic device for automated biology. *Lab on a Chip*, 11(9):1612–1619.

Foudeh, A. M., Didar, T. F., Veres, T., and Tabrizian, M. (2012). Microfluidic designs and techniques using lab-on-a-chip devices for pathogen detection for point-of-care diagnostics. *Lab on a Chip*, 12(18):3249–3266.

Friedland, A. E., Lu, T. K., Wang, X., Shi, D., Church, G., and Collins, J. J. (2009). Synthetic gene networks that count. *Science*, 324(5931):1199–1202.

Galdzicki, M., Wilson, M., Rodriguez, C. A., Pocock, M. R., Oberortner, E., Adam, L., Adler, A., Anderson, J. C., Beal, J., Cai, Y., et al. (2012). Synthetic biology open language (sbol) version 1.1. 0. `http://dspace.mit.edu/handle/1721.1/73909`. Accessed: 2014-06-17.

Gibson, D. G., Young, L., Chuang, R.-Y., Venter, J. C., Hutchison, C. A., and Smith, H. O. (2009). Enzymatic assembly of dna molecules up to several hundred kilobases. *Nature methods*, 6(5):343–345.

Goldberg, M. (2013). Biofab: Applying moore's law to dna synthesis. *Industrial Biotechnology*, 9(1):10–12.

Gómez-Sjöberg, R., Leyrat, A. A., Pirone, D. M., Chen, C. S., and Quake, S. R. (2007). Versatile, fully automated, microfluidic cell culture system. *Analytical chemistry*, 79(22):8557–8563.

Goñi-Moreno, A. and Amos, M. (2012). A reconfigurable nand/nor genetic logic gate. *BMC systems biology*, 6(1):126.

Gupta, K., Kim, D.-H., Ellison, D., Smith, C., Kundu, A., Tuan, J., Suh, K.-Y., and Levchenko, A. (2010). Lab-on-a-chip devices as an emerging platform for stem cell biology. *Lab on a Chip*, 10(16):2019–2031.

Gupta, S., Bram, E. E., and Weiss, R. (2013). Genetically programmable pathogen sense and destroy. *ACS synthetic biology*, 2(12):715–723.

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

Haeberle, S. and Zengerle, R. (2007). Microfluidic platforms for lab-on-a-chip applications. *Lab on a Chip*, 7(9):1094–1110.

Ham, T. S., Lee, S. K., Keasling, J. D., and Arkin, A. P. (2008). Design and construction of a double inversion recombination switch for heritable sequential genetic memory. *PLoS One*, 3(7):e2815.

Hamon, M. and Hong, J. W. (2013). New tools and new biology: Recent miniaturized systems for molecular and cellular biology. *Molecules and cells*, 36(6):485–506.

Hassoun, S. and Sasao, T. (2002). *Logic Synthesis and Verification*. The Springer International Series in Engineering and Computer Science. Springer US.

Hayes, J. P. (1993). *Digital Logic Design*. Addison Wesley.

Haynes, K. A., Broderick, M. L., Brown, A. D., Butner, T. L., Dickson, J. O., Harden, W. L., Heard, L. H., Jessen, E. L., Malloy, K. J., Ogden, B. J., et al. (2008). Engineering bacteria to solve the burnt pancake problem. *Journal of biological engineering*, 2(8):1–12.

Hillson, N. J., Rosengarten, R. D., and Keasling, J. D. (2012). j5 dna assembly design automation software. *ACS synthetic biology*, 1(1):14–21.

Hong, J. W., Chen, Y., Anderson, W. F., and Quake, S. R. (2006). Molecular biology on a microfluidic chip. *Journal of Physics: Condensed Matter*, 18(18):S691.

Hu, K., Dinh, T. A., Ho, T.-Y., and Chakrabarty, K. (2014). Control-layer optimization for flow-based mvlsi microfluidic biochips. In *IEEE 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES),*, pages 1–10.

Huang, H. and Densmore, D. (2014a). Fluigi: Microfluidic device synthesis for synthetic biology. *ACM journal on emerging technologies in computing systems*, 11(3):26:1–26:19.

Huang, H. and Densmore, D. (2014b). Integration of microfluidics into the synthetic biology design flow. *Lab Chip*, 14:3459–3474.

Huang, M. C., Ye, H., Kuan, Y. K., Li, M.-H., and Ying, J. Y. (2009). Integrated two-step gene synthesis in a microfluidic device. *Lab on a Chip*, 9(2):276–285.

Inamdar, N. K. and Borenstein, J. T. (2011). Microfluidic cell culture models for tissue engineering. *Current opinion in biotechnology*, 22(5):681–689.

Jang, S. S., Oishi, K. T., Egbert, R. G., and Klavins, E. (2012). Specification and simulation of synthetic multicelled behaviors. *ACS synthetic biology*, 1(8):365–374.

Kersaudy-Kerhoas, M., Amalou, F., Che, A., Kelly, J., Liu, Y., Desmulliez, M., and Shu, W. (2014). Validation of a fully integrated platform and disposable microfluidic chips enabling parallel purification of genome segments for assembly. *Biotechnology and bioengineering*, 111(8):1627–1637.

Khalil, A. S. and Collins, J. J. (2010). Synthetic biology: applications come of age. *Nature Reviews Genetics*, 11(5):367–379.

Khalil, A. S., Lu, T. K., Bashor, C. J., Ramirez, C. L., Pyenson, N. C., Joung, J. K., and Collins, J. J. (2012). A synthetic biology framework for programming eukaryotic transcription functions. *Cell*, 150(3):647–658.

Kim, J., Taylor, D., Agrawal, N., Wang, H., Kim, H., Han, A., Rege, K., and Jayaraman, A. (2012). A programmable microfluidic cell array for combinatorial drug screening. *Lab on a chip*, 12(10):1813–1822.

Kirby, B. (2010). *Micro- and Nanoscale Fluid Mechanics: Transport in Microfluidic Devices*. Cambridge University Press.

Kittleson, J. T., Wu, G. C., and Anderson, J. C. (2012). Successes and failures in modular genetic engineering. *Current opinion in chemical biology*, 16(3):329–336.

Klumpp, S., Zhang, Z., and Hwa, T. (2009). Growth rate-dependent global effects on gene expression in bacteria. *Cell*, 139(7):1366–1375.

Kong, D. S., Carr, P. A., Chen, L., Zhang, S., and Jacobson, J. M. (2007). Parallel gene synthesis in a microfluidic device. *Nucleic acids research*, 35(8):e61.

Kosuri, S., Eroshenko, N., LeProust, E., Super, M., Way, J., Li, J. B., and Church, G. M. (2010). A scalable gene synthesis platform using high-fidelity dna microchips. *Nature biotechnology*, 28(12):1295.

Lecault, V., White, A. K., Singhal, A., and Hansen, C. L. (2012). Microfluidic single cell analysis: from promise to practice. *Current opinion in chemical biology*, 16(3):381–390.

Lee, C.-C., Snyder, T. M., and Quake, S. R. (2010). A microfluidic oligonucleotide synthesizer. *Nucleic acids research*, 38(8):2514–2521.

Leguia, M., Brophy, J. A., Densmore, D., Asante, A., and Anderson, J. C. (2013). 2ab assembly: a methodology for automatable, high-throughput assembly of standard biological parts. *Journal of biological engineering*, 7(1):1–16.

Lei, K. F. (2012). Microfluidic systems for diagnostic applications a review. *Journal of laboratory automation*, 17(5):330–347.

Lin, B. and Levchenko, A. (2012). Microfluidic technologies for studying synthetic circuits. *Current opinion in chemical biology*, 16(3):307–317.

Lin, F., Saadi, W., Rhee, S. W., Wang, S.-J., Mittal, S., and Jeon, N. L. (2004). Generation of dynamic temporal and spatial concentration gradients using microfluidic devices. *Lab on a Chip*, 4(3):164–167.

Liu, W., Li, L., Wang, X., Ren, L., Wang, X., Wang, J., Tu, Q., Huang, X., and Wang, J. (2010). An integrated microfluidic system for studying cell-microenvironmental interactions versatilely and dynamically. *Lab on a Chip*, 10(13):1717–1724.

Locke, J. C. and Elowitz, M. B. (2009). Using movies to analyse gene circuit dynamics in single cells. *Nature Reviews Microbiology*, 7(5):383–392.

Long, Z., Nugent, E., Javer, A., Cicuta, P., Sclavi, B., Lagomarsino, M. C., and Dorfman, K. D. (2013). Microfluidic chemostat for measuring single cell dynamics in bacteria. *Lab on a Chip*, 13(5):947–954.

Lu, T., Ferry, M., Weiss, R., and Hasty, J. (2008). A molecular noise generator. *Physical biology*, 5(3):036006.

Lu, T. K. (2010). Engineering scalable biological systems. *Bioengineered bugs*, 1(6):378–384.

Lu, T. K. and Collins, J. J. (2007). Dispersing biofilms with engineered enzymatic bacteriophage. *Proceedings of the National Academy of Sciences*, 104(27):11197–11202.

Lu, T. K. and Collins, J. J. (2009). Engineered bacteriophage targeting gene networks as adjuvants for antibiotic therapy. *Proceedings of the National Academy of Sciences*, 106(12):4629–4634.

Lux, M. W., Bramlett, B. W., Ball, D. A., and Peccoud, J. (2012). Genetic design automation: engineering fantasy or scientific renewal? *Trends in biotechnology*, 30(2):120–126.

Ma, S., Tang, N., and Tian, J. (2012). Dna synthesis, assembly and applications in synthetic biology. *Current opinion in chemical biology*, 16(3):260–267.

Macía, J., Posas, F., and Solé, R. V. (2012). Distributed computation: the new wave of synthetic biology devices. *Trends in biotechnology*, 30(6):342–349.

Madsen, C., Myers, C. J., Patterson, T., Roehner, N., Stevens, J. T., and Winstead, C. (2012). Design and test of genetic circuits using ibiosim. *IEEE Design & Test of Computers*, 29(3):32–39.

Mark, D., Haeberle, S., Roth, G., von Stetten, F., and Zengerle, R. (2010). Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Chemical Society Reviews*, 39(3):1153–1182.

McDaniel, J., Baez, A., Crites, B., Tammewar, A., and Brisk, P. (2013). Design and verification tools for continuous fluid flow-based microfluidic devices. In *18th Asia and South Pacific Design Automation Conference (ASP-DAC 2013)*, pages 219–224.

McDaniel, J., Parker, B., and Brisk, P. (2014). Simulated annealing-based placement for microfluidic large scale integration (mlsi) chips. In *IEEE 2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC),*, pages 1–6.

Melin, J. and Quake, S. R. (2007). Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annual review of biophysics and biomolecular structure*, 36:213–231.

Micheli, G. D. (1994). *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education.

Milias-Argeitis, A., Summers, S., Stewart-Ornstein, J., Zuleta, I., Pincus, D., El-Samad, H., Khammash, M., and Lygeros, J. (2011). In silico feedback for in vivo regulation of a gene expression circuit. *Nature biotechnology*, 29(12):1114–1116.

Minhass, W. H., Pop, P., Madsen, J., and Blaga, F. S. (2012). Architectural synthesis of flow-based microfluidic large-scale integration biochips. In *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*, pages 181–190. ACM.

Minhass, W. H., Pop, P., Madsen, J., and Ho, T.-Y. (2013). Control synthesis for the flow-based microfluidic large-scale integration biochips. In *18th Asia and South Pacific Design Automation Conference (ASP-DAC 2013)*, pages 205–212.

Mondragón-Palomino, O., Danino, T., Selimkhanov, J., Tsimring, L., and Hasty, J. (2011). Entrainment of a population of synthetic genetic oscillators. *Science*, 333(6047):1315–1319.

Moon, T. S., Lou, C., Tamsir, A., Stanton, B. C., and Voigt, C. A. (2012). Genetic programs constructed from layered logic gates in single cells. *Nature*, 491(7423):249–253.

Mosadegh, B., Kuo, C.-H., Tung, Y.-C., Torisawa, Y.-s., Bersano-Begey, T., Tavana, H., and Takayama, S. (2010). Integrated elastomeric components for autonomous regulation of sequential and oscillatory flow switching in microfluidic devices. *Nature physics*, 6(6):433–437.

Nestor, J. (2007). Cadapplets animations of vlsi cad algorithms. `http://workbench.lafayette.edu/~nestorj/cadapplets`. Accessed: 2015-10-24.

Nguyen, T. V., Duncan, P. N., Ahrar, S., and Hui, E. E. (2012). Semi-autonomous liquid handling via on-chip pneumatic digital logic. *Lab on a Chip*, 12(20):3991–3994.

Nissim, L. and Bar-Ziv, R. H. (2010). A tunable dual-promoter integrator for targeting of cancer cells. *Molecular systems biology*, 6(1):444.

Oh, K. W., Lee, K., Ahn, B., and Furlani, E. P. (2012). Design of pressure-driven microfluidic networks using electric circuit analogy. *Lab on a Chip*, 12(3):515–545.

Pedersen, M. and Phillips, A. (2009). Towards programming languages for genetic engineering of living cells. *Journal of the Royal Society Interface*, 6(Suppl 4):S437–S450.

Peralta-Yahya, P. P., Zhang, F., Del Cardayre, S. B., and Keasling, J. D. (2012). Microbial engineering for the production of advanced biofuels. *Nature*, 488(7411):320–328.

Prindle, A., Samayoa, P., Razinkov, I., Danino, T., Tsimring, L. S., and Hasty, J. (2012). A sensing array of radically coupled genetic 'biopixels'. *Nature*, 481(7379):39–44.

Purcell, O., Savery, N. J., Grierson, C. S., and di Bernardo, M. (2010). A comparative analysis of synthetic genetic oscillators. *Journal of The Royal Society Interface*, 7(52):1503–1524.

Purnick, P. E. and Weiss, R. (2009). The second wave of synthetic biology: from modules to systems. *Nature reviews. Molecular cell biology*, 10(6):410–422.

Regot, S., Macia, J., Conde, N., Furukawa, K., Kjellén, J., Peeters, T., Hohmann, S., de Nadal, E., Posas, F., and Solé, R. (2011). Distributed biological computation with multicellular engineered networks. *Nature*, 469(7329):207–211.

Ro, D.-K., Paradise, E. M., Ouellet, M., Fisher, K. J., Newman, K. L., Ndungu, J. M., Ho, K. A., Eachus, R. A., Ham, T. S., Kirby, J., et al. (2006). Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature*, 440(7086):940–943.

Ruder, W. C., Lu, T., and Collins, J. J. (2011). Synthetic biology moving into the clinic. *Science*, 333(6047):1248–1252.

Salis, H. M., Mirsky, E. A., and Voigt, C. A. (2009). Automated design of synthetic ribosome binding sites to control protein expression. *Nature biotechnology*, 27(10):946–950.

Schmidt, M.-J., Gasiūnaitė, L., French, C., Hale, A., and Gallagher, T. (2013). Multiplex dna assmembly technology. http://www.genabler.com/media/1353/postersb60_genabler_3-07-13.pdf.

Sedgewick, R. and Wayne, K. (2015). Geometric intersections. `http://algs4.cs.princeton.edu/93intersection/`. Accessed: 2015-10-24.

Shankar, S. and Pillai, M. R. (2011). Translating cancer research by synthetic biology. *Molecular BioSystems*, 7(6):1802–1810.

Shetty, R. (2013). The latency of gene synthesis - 2013 update. http://blog.ginkgobioworks.com/2013/08/17/gene-synthesis-latency/.

Shetty, R. P., Endy, D., and Knight Jr, T. F. (2008). Engineering biobrick vectors from biobrick parts. *Journal of biological engineering*, 2(1):1–12.

Sia, S. K. and Whitesides, G. M. (2003). Microfluidic devices fabricated in poly (dimethylsiloxane) for biological studies. *Electrophoresis*, 24(21):3563–3576.

Simpson, M. L., Cox, C. D., and Sayler, G. S. (2003). Frequency domain analysis of noise in autoregulated gene circuits. *Proceedings of the National Academy of Sciences*, 100(8):4551–4556.

Siuti, P., Yazbek, J., and Lu, T. K. (2013). Synthetic circuits integrating logic and memory in living cells. *Nature biotechnology*, 31(5):448–452.

Sohka, T., Heins, R. A., Phelan, R. M., Greisler, J. M., Townsend, C. A., and Ostermeier, M. (2009). An externally tunable bacterial band-pass filter. *Proceedings of the National Academy of Sciences*, 106(25):10135–10140.

Sollier, E., Murray, C., Maoddi, P., and Di Carlo, D. (2011). Rapid prototyping polymers for microfluidic devices and high pressure injections. *Lab on a Chip*, 11(22):3752–3765.

Squires, T. M. and Quake, S. R. (2005). Microfluidics: Fluid physics at the nanoliter scale. *Reviews of modern physics*, 77(3):977.

Stanton, B. C., Nielsen, A. A., Tamsir, A., Clancy, K., Peterson, T., and Voigt, C. A. (2014). Genomic mining of prokaryotic repressors for orthogonal logic gates. *Nature chemical biology*, 10(2):99–105.

Stroock, A. D., Dertinger, S. K., Ajdari, A., Mezić, I., Stone, H. A., and Whitesides, G. M. (2002). Chaotic mixer for microchannels. *Science*, 295(5555):647–651.

Su, F. and Chakrabarty, K. (2005). Design of fault-tolerant and dynamically-reconfigurable microfluidic biochips. In *IEEE Proceedings on Design, Automation and Test in Europe, 2005.*, pages 1202–1207. IEEE.

Su, F., Chakrabarty, K., and Fair, R. B. (2006). Microfluidics-based biochips: technology issues, implementation platforms, and design-automation challenges. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(2):211–223.

Sung, J. H. and Shuler, M. L. (2010). In vitro microscale systems for systematic drug toxicity study. *Bioprocess and biosystems engineering*, 33(1):5–19.

Tamsir, A., Tabor, J. J., and Voigt, C. A. (2011). Robust multicellular computing using genetically encoded nor gates and chemical/wires/'. *Nature*, 469(7329):212–215.

Taylor, R., Falconnet, D., Niemistö, A., Ramsey, S., Prinz, S., Shmulevich, I., Galitski, T., and Hansen, C. (2009). Dynamic analysis of mapk signaling using a high-throughput microfluidic single-cell imaging platform. *Proceedings of the National Academy of Sciences*, 106(10):3758–3763.

Thies, W., Urbanski, J. P., Thorsen, T., and Amarasinghe, S. (2008). Abstraction layers for scalable microfluidic biocomputing. *Natural Computing*, 7(2):255–275.

Thies, W., Urbanski, J. P., Thorsen, T., and Amarasinghe, S. (2009). Programable microfluidics. `http://groups.csail.mit.edu/cag/biostream/`. Accessed: 2014-06-17.

Thorsen, T., Maerkl, S. J., and Quake, S. R. (2002). Microfluidic large-scale integration. *Science*, 298(5593):580–584.

Toettcher, J. E., Gong, D., Lim, W. A., and Weiner, O. D. (2011a). Light-based feedback for controlling intracellular signaling dynamics. *Nature methods*, 8(10):837–839.

Toettcher, J. E., Voigt, C. A., Weiner, O. D., and Lim, W. A. (2011b). The promise of optogenetics in cell biology: interrogating molecular circuits in space and time. *Nature methods*, 8(1):35–38.

Tseng, K.-H., You, S.-C., Liou, J.-Y., and Ho, T.-Y. (2013). A top-down synthesis methodology for flow-based microfluidic biochips considering valve-switching minimization. In *Proceedings of the 2013 ACM international symposium on physical design*, pages 123–129. ACM.

Unger, M. A., Chou, H.-P., Thorsen, T., Scherer, A., and Quake, S. R. (2000). Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science*, 288(5463):113–116.

Urbanski, J. P., Thies, W., Rhodes, C., Amarasinghe, S., and Thorsen, T. (2006). Digital microfluidics using soft lithography. *Lab on a Chip*, 6(1):96–104.

Vaidyanathan, P., Der, B., Bhatia, S., Roehner, N., Silva, R., Voigt, C., and Densmore, D. (2015). A framework for genetic logic synthesis. *Proceedings of the IEEE*, PP(99):1–12.

Voigt, C. A. (2006). Genetic parts to program bacteria. *Current opinion in biotechnology*, 17(5):548–557.

Waldbaur, A., Kittelmann, J., Radtke, C. P., Hubbuch, J., and Rapp, B. E. (2013). Microfluidics on liquid handling stations (μf-on-lhs): an industry compatible chip interface between microfluidics and automated liquid handling stations. *Lab on a Chip*, 13(12):2337–2343.

Wang, B., Kitney, R. I., Joly, N., and Buck, M. (2011). Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology. *Nature communications*, 2:508.

Wang, C. J., Bergmann, A., Lin, B., Kim, K., and Levchenko, A. (2012). Diverse sensitivity thresholds in dynamic signaling responses by social amoebae. *Science signaling*, 5(213):ra17.

Wang, H. H., Isaacs, F. J., Carr, P. A., Sun, Z. Z., Xu, G., Forest, C. R., and Church, G. M. (2009). Programming cells by multiplex genome engineering and accelerated evolution. *Nature*, 460(7257):894–898.

Weaver, J. A., Melin, J., Stark, D., Quake, S. R., and Horowitz, M. A. (2010). Static control logic for microfluidic devices using pressure-gain valves. *Nature Physics*, 6(3):218–223.

Weber, E., Engler, C., Gruetzner, R., Werner, S., and Marillonnet, S. (2011). A modular cloning system for standardized assembly of multigene constructs. *PLoS one*, 6(2):e16765.

Weibel, D. B., DiLuzio, W. R., and Whitesides, G. M. (2007). Microfabrication meets microbiology. *Nature Reviews Microbiology*, 5(3):209–218.

Weste, N. and Harris, D. (2011). *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison Wesley Publishing Company Incorporated.

Westfall, P. J. and Gardner, T. S. (2011). Industrial fermentation of renewable diesel fuels. *Current opinion in biotechnology*, 22(3):344–350.

Whitesides, G. M. (2006). The origins and the future of microfluidics. *Nature*, 442(7101):368–373.

Xia, B., Bhatia, S., Bubenheim, B., Dadgar, M., Densmore, D., and Anderson, J. C. (2011). Developers and users guide to clotho v2. 0 a software platform for the creation of synthetic biological systems. *Methods in enzymology*, 498:97–135.

Xie, Z., Wroblewska, L., Prochazka, L., Weiss, R., and Benenson, Y. (2011). Multi-input rnai-based logic circuit for identification of specific cancer cells. *Science*, 333(6047):1307–1311.

Yin, H. and Marshall, D. (2012). Microfluidics for single cell analysis. *Current opinion in biotechnology*, 23(1):110–119.

Yu, F., Horowitz, M. A., and Quake, S. R. (2013). Microfluidic serial digital to analog pressure converter for arbitrary pressure generation and contamination-free flow control. *Lab on a Chip*, 13(10):1911–1918.

Zare, R. N. and Kim, S. (2010). Microfluidic platforms for single-cell analysis. *Annual review of biomedical engineering*, 12:187–201.

Zhang, C., Xu, J., Ma, W., and Zheng, W. (2006). Pcr microfluidic devices for dna amplification. *Biotechnology advances*, 24(3):243–284.

Zhang, Q. and Austin, R. H. (2012). Applications of microfluidics in stem cell biology. *BioNanoScience*, 2(4):277–286.

Zhang, W., Lin, S., Wang, C., Hu, J., Li, C., Zhuang, Z., Zhou, Y., Mathies, R. A., and Yang, C. J. (2009). Pmma/pdms valves and pumps for disposable microfluidics. *Lab on a Chip*, 9(21):3088–3094.