Microfluidic Molecular Processors for Computation and Analysis

by

William Henri Grover

B.S. (University of Tennessee, Knoxville) 1999

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Chemistry

in the

GRADUATE DIVISION
of the
UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor Richard A. Mathies, Chair
Professor A. Paul Alivisatos
Professor Luke P. Lee

Spring 2006

The dissertation of William Henri Grover is approved:

Chair Date

Date

Date

University of California, Berkeley

Spring 2006

Microfluidic Molecular Processors for Computation and Analysis

by

William Henri Grover

Abstract

Microfluidic Molecular Processors for Computation and Analysis

by

William Henri Grover

Doctor of Philosophy in Chemistry

University of California, Berkeley

Professor Richard A. Mathies, Chair

Principles from microfabrication, computer science, and molecular biology are combined to develop *microfluidic processors* for computational and analytical applications. These microprocessors utilize monolithic membrane valves that are combined to form pumps, routers, and complex valve networks. The processors are fabricated by sandwiching a polydimethylsiloxane (PDMS) membrane between two etched glass wafers. A valve is formed where a channel discontinuity in one wafer lies across the PDMS membrane from an etched displacement chamber. Depressurizing the chamber pulls the membrane away from the discontinuous channel and opens the valve. Three independently-controlled valves in series form a diaphragm pump. The valves have < 10 nL dead volumes and seal against 75 kPa pressure; the pumps are self-priming and pump 1–100 nL/s. Valves are fabricated in dense arrays and actuated in parallel via integrated pneumatic channels. This technology has facilitated the development of lab-on-a-chip devices for pathogen detection, nanoliter-scale DNA sequencing, and

extraterrestrial amino acid analysis.

My focus was on the development of microfluidic molecular processors that use single DNA bases (SNPs) to represent binary bits in a DNA computation. The processor includes selectable chambers containing magnetic capture beads. Fluorescent input DNA containing polymorphic bases is recirculated on-chip through selected capture beads; perfectly-complementary DNA is captured by hybridization to bead-immobilized oligonucleotides. Input DNA remaining after several capture/release steps provides the answer to a Boolean satisfiability problem. The improved capture kinetics, 92% step-to-step transfer efficiency, and single-base specificity enabled by microfluidics make possible this first microfluidic DNA computer. Prospects for performing analyses like haplotyping of genomic DNA with this processor are also presented.

I also show that microfluidic processors can be programmed to control their own operation using pneumatic logical circuits. Valve networks are demonstrated that function as latching valves and as demultiplexers. Latching valve circuits use 120 ms vacuum/pressure pulses to hold valves open or closed, and on-chip demultiplexers require only n pneumatic inputs to control $2^{(n-1)}$ independent latching valves. These technologies enable the independent control of large numbers of on-chip valves. This research provides the foundations for programmable microfluidic processors or automatons, generic microfluidic devices that can perform arbitrary computations or analyses.

Professor Richard A. Mathies
Dissertation Committee Chair

For the best parents in the world

Bill and Julie Grover

with love and gratitude

Contents

List of Figures

List of Tables

Acknowledgments

Thanks first and foremost to Prof. Rich Mathies for making this journey possible, for giving me the freedom to explore the landscape of my project and for keeping me focused when I strayed a little too far. And thanks to Mary Hammond for always finding time in her busy day to give me a hand with the million little details that made this work possible. Grad school is always a daunting process, but I know that Rich and Mary were always on my side, speaking well of me to others and helping me through.

Thanks to the members of my qualifying exam committee, Professors Richard Karp, Peidong Yang, Birgitta Whaley, and Marcin Majda. Thanks also to the members of my thesis committee, Professors Luke Lee and Paul Alivisatos, for reading a chemistry dissertation that goes from defecating ducks to player pianos, and that's only the first ten pages.

In hindsight, I chose to attend the University of Tennessee and the University of California mostly because of wonderful experiences I'd had at both schools as a high school student. I attended the Tennessee Governor's School for the Sciences at UT in 1994, and I left with such a positive impression of UT that I enrolled there as an undergraduate in 1995 and returned to Gov School as a Resident Assistant in 1998 and 1999. I was also fortunate enough to represent Tennessee in the Department of Energy High School Honors Program in the Life Sciences at UC Berkeley in 1995. I remembered Berkeley so fondly from those two weeks, and the opportunity to return

here for grad school was like a dream come true. These programs show students that there's a world outside of their hometowns, and great opportunities are out there if you're willing to look. Unfortunately, these programs are often the first to succumb to budget cuts. To Professors Charles Lane and Jeffery Kovac at UT, Eileen Engel and the late Prof. Glenn Seaborg at LBNL and UC, and everyone else who played a part in making those programs happen, I'm here writing this because of what you did.

My earliest and fondest memories of my time in Berkeley always involve Mike Tauber, Judy Kim, and Dave McCamant, three Ramanites who took a young DNAer into their lab and under their wings. Now a day rarely goes by that I don't rely on something that Mike, Judy, or Dave taught me during those early years. Later on, Brian Paegel and Eric Lagally helped me in countless ways, professionally and personally. Mike, Judy, Dave, Brian, and Eric, you've set the bar pretty high for those of us who followed in your footsteps. My respect for you as scientists is eclipsed only by my fondness for you as friends.

I had the privilege of teaching Chemistry 125 in the fall of 2001 with Dr. Benjamin Boussert. Ben was the perfect teaching companion, skillful, selfless, and hard-working. His death along with Dr. Jason Choy and Dr. Giulia Adesso in a car accident on July 16, 2005 left a terrible void at Berkeley and robbed future scientists of the privilege of learning chemistry from a natural teacher.

All of my colleagues in the Mathies Group are outstanding scientists, but I'm

amazed by how many of them are also "civil scientists," involved in the positive role that science can play in society. Few things make me more hopeful for the future than the realization that these outstanding men and women will leave Berkeley with both a first-rate science education and a passion for using that education to bridge the gap between the scientist and the citizen. They remind me of Edward R. Murrow's words spoken in 1958 about television but applicable to so many human endeavors, especially science:

> This instrument can teach, it can illuminate; yes, and it can even inspire. But it can do so only to the extent that humans are determined to use it to those ends. Otherwise it is merely wires and lights in a box. There is a great and perhaps decisive battle to be fought against ignorance, intolerance and indifference.

I'm proud to have colleagues who are on the front lines of that battle.

Over the last year, I've had the immense pleasure of working with first-year Biophysics graduate student Erik Jensen. My "computing project" combined computer science, engineering, chemistry, molecular biology, and three or four other fields in such a weird mix that I doubted I'd ever find anyone interested in continuing it once I left. Then along came Erik, who has not only the perfect background for the project but also an inventive streak that will drive the project forward in ways that I can't even imagine. Erik, working with you has been the intellectual highlight of my time at Berkeley. I look forward to seeing incredible research from you over the coming years.

To my parents: I think I've finally run out of school. Hopefully that means that

I'll stop asking you for money some time in the near future. I got through the last seven years only because of how well you prepared me during the previous twenty-two years. I'll be blessed if one day I can be half as good a parent as you both have been to me.

And finally, to Alison: Grad school was worthwhile because I found you; everything else good that happened here was just gravy. You've supported me so much through these years that in all fairness there should be two names on this thesis instead of just one. I know I couldn't have done this without you. And whenever I'm uncertain about the future, knowing that we'll face it together makes it okay—the important stuff is taken care of—the rest are just details.

By some chance, here they are, all on this earth; and who shall ever tell the sorrow of being on this earth, lying, on quilts, on the grass, in a summer evening, among the sounds of the night.

—*James Agee,* Knoxville: Summer 1915

Chapter 1

Introduction

> *Science is what we understand well enough to explain to a computer. Art is everything else we do.*
>
> —*Donald E. Knuth* [1]

The fields of molecular biology and computer science arguably represent the most significant scientific accomplishments of the last fifty years. Certainly no fields of human inquiry have evolved as quickly as molecular biology and computer science. These subjects went from obscurity to ubiquity in a *single lifetime*: one of the scientists who discovered the structure of DNA went on to direct the effort to sequence the human genome, and one of the designers of the ENIAC lived to see desktop personal computers in every office and classroom. Today, molecular biology and computer science continue to affect us on the most fundamental levels: how we see ourselves, and how we communicate with each other.

It is therefore not surprising that research that *combines* aspects of both molecular

biology and computer science has produced some of both fields' most interesting and promising results. Indeed, the line between the two fields seems to be disappearing with time: computers now exploit the concepts of evolution when solving problems with genetic algorithms, and biologists use sophisticated computer programs to sift through terabytes of DNA sequence data. Two specific areas of research at the intersection between computer science and molecular biology are the focus of this dissertation. The first, the field of *DNA computing*, seeks to build computers that use DNA to both store and operate upon digital information. The second, the field of *microfluidic analysis devices*, uses technologies borrowed from the computer chip making industry to build devices that operate upon and analyze biological or chemical samples.

The research presented in this dissertation represents an attempt to answer a basic question: *how do you compute in a microfluidic device?* This simple question has a surprising number of different answers. In Chapter 2, the transistors of a microfluidic computer—monolithic membrane valves—are presented and characterized. Like transistors in silicon microprocessors, these valves can be fabricated in dense arrays and actuated serially or in large parallel arrays. They form the technological foundation of the rest of the work presented in this dissertation. In Chapter 3, the valves are used to create a microfluidic DNA computer. Just as a conventional microprocessor routes electric charges between operations, the microfluidic computer routes microliter-scale volumes of DNA through various hybridization-based capture-

and-release operations. This device represents the first full-scale demonstration of a microfluidic DNA computer, and the improved capture efficiency made possible the first use of single DNA bases as binary bits in a hybridization-based computation. In Chapter 4, logical circuits of the monolithic membrane valves are designed and demonstrated. Just as transistors are arranged into logical circuits in conventional microprocessors, microfluidic valves can be arranged into pneumatic logical circuits and used to control other valves. The multiplexed latching valve circuits demonstrated in Chapter 4 are an example of the huge variety of on-chip logical circuits that can be fabricated using valves. These structures can greatly reduce the number of off-chip controllers required to operate a microfluidic device. In Chapter 5, a variety of other technologies for designing and controlling microfluidic analysis devices are presented. Finally, in Chapter 6, prospects for future microfluidic computers are discussed. Taken as a whole, this work represents the technological foundation for a *programmable fluidic microprocessor*, a generic, programmable device that can perform countless different computations or analyses simply by applying different inputs and running different programs. Such a device could revolutionize the analytical microdevice community in the same way that programmable computers revolutionized the fledgling computer industry over fifty years ago. It would represent the ultimate answer to the question, *how do you compute in a microfluidic device?*

In this Introduction, the historical and technological foundations of this research are discussed. The history of the intersection between computers and biology begins

with the mechanical automata of the eighteenth century, which demonstrated that living systems could be though of in mechanical and programmable terms. Early pneumatic control systems in player pianos are also discussed briefly because they offer insights into how modern microfluidic devices can be controlled by similar pneumatic logical principles. John von Neumann's remarkably insightful proposals about biological and evolutionary computers are presented as the theoretical foundation for the biomolecular computers presented in this dissertation. An analysis of Michael Conrad's "tradeoff principle" suggests that microfluidic DNA computers could combine evolvability, programmability, and efficiency in ways that existing electronic or DNA computers cannot. The field of cellular automata is then discussed briefly as background for later sections about both self-assembled DNA nanostructures and automaton-like programmable microfluidic processors. Finally, a review of relevant experiments in DNA computation is presented, from Adleman's seminal computation to early proposals for microfluidic DNA computers.

1.1 Programmable life: Mechanical automata

Remarkably, one of the first programmable machines was a biological computer of sorts. In 1738, more than two hundred years before the first programmable computers were demonstrated, the Frenchman Jacques de Vaucanson built and demonstrated a programmable mechanical duck. Though whimsical in hindsight, Vaucanson's duck had a profound influence on the development of programmable computers,

both biomolecular and otherwise.

A drawing of Jacques de Vaucanson's *Canard Digérateur* or "Digesting Duck" is shown in Figure 1.1. The life-size copper duck was an exhaustive copy of a living

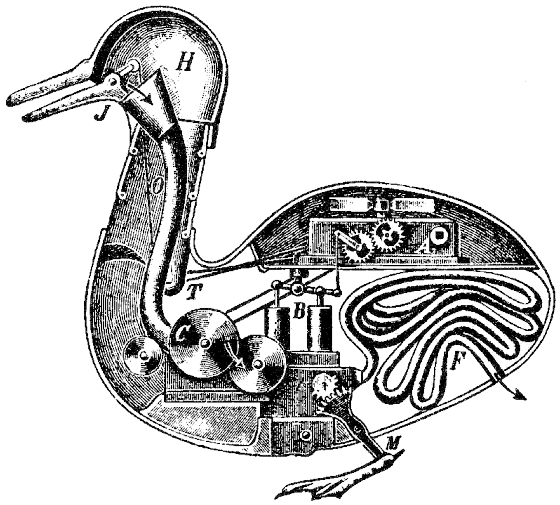


Figure 1.1: Illustration of Jacques de Vaucanson's "Digesting Duck." From the public domain.

duck; it could "drink, muddle the water with its beak, quack, rise, and settle back on its legs" [2]. Vaucanson wrote that the duck "stretches out his neck to go and take the grain from the hand, he swallows it, digests it, and excretes it, once digested, through the normal channels" [3]. The Digesting Duck enthralled paying audiences as a concrete representation of Enlightenment-era ideas about men and machines [2].

The Digesting Duck's lifelike motions were controlled using an intricate mechanism actuated by a large rotating drum studded with pins [3]. Similar mechanisms had been used to encode songs in bell towers and music boxes since the eighth century [4],

but in Vaucanson's duck they were put to a novel use: mechanically programming the behavior of a "living organism." Vaucanson reduced a living thing to an *automaton*, a programmable machine. The historian Jacques Perriault goes even further and describes the Digesting Duck as a sort of primitive programmable computer:

> The mechanism is completely in charge, automated from start to finish. There is a control unit which directs the various mechanical elements which, in the eighteenth century, would have been connecting rods, cams, and pistons. Just as the arithmetical operation $a + b$ is rewritten in a sequence of transfers and counts [in a computer program], so the analysis of the duck's waddle is to be found in the sequencing of these various elements. [The pins on the duck's rotating drum] represent an *encoding* of the rules of the algorithm. Today they would be called *instructions*. As for the drum, it is an implement that one would be quite wrong to consider a novelty: it is a *programme* [5].

By reducing a living duck to a programmable machine, Vaucanson set the stage for many other innovations that blurred the line between biology and computation, including DNA computing. He also demonstrated that programmable mechanisms could replicate even the most complex phenomena (in this case, the behavior of a living duck). But the duck's mechanical processor and pinned-drum program were too complex to be of practical use in later programmable machines. Instead, *pneumatic logic* and *punched-paper programs* were used as simple and effective tools in the first mass-produced programmable machine: the player piano.

1.2 Programmable air: Player pianos and pneumatic logic

From their introduction around 1900 to their decline in the Great Depression, 2.5 million pneumatic player pianos were sold in the United States [4]. Two technological innovations were responsible for their immense success: a scalable, multiplexed actuation principle (pneumatics), and an inexpensive, versatile controller (pneumatic logic and punched paper). The principles of programmable pneumatic logic employed in the player piano form the historical and technical foundation of the microfluidic pneumatic logic circuits discussed in later sections.

Pneumatic logic and punched paper programs were developed in response to shortcomings in the pinned-drum program used in the Digesting Duck. Musical instruments, looms, and other machines that used pinned-drum programs required a new drum for each new song or pattern [4]. Delicate and hand-made, the drums were unsuitable for these purposes. In 1804, Frenchman Joseph-Marie Jacquard used punched cards to program weaving patterns in his Jacquard Loom [3]. The presence or absence of a punched hole was detected by sliding rods pressed against the card, and the position of these rods was then used to set the desired pattern in the loom. The inexpensive cards could be made by machine and duplicated easily, and Jacquard's loom was an immediate success. But early (non-pneumatic) attempts at controlling pianos with punched paper were unsuccessful, in part because playing a piano requires much

more force than arranging strands of thread, and any mechanism powerful enough to play a piano would also shred the punched paper [4]. An *amplifier* was needed for sensing small holes in the delicate paper and driving the hammer against the string.

In 1896, American inventor Edwin Scott Votey patented the "Pianola," the first commercially successful player piano [4]. The Pianola used pneumatics to both read holes in the punched paper program and operate the piano. A schematic showing the mechanism that operates a single key in the Pianola is shown in Figure 1.2A. The mechanism contains two pneumatically-actuated valves per key, a *primary valve* consisting of two pistons (brown) connected to a flexible membrane (green), and a secondary valve with a single piston connected to a membrane. These valves are connected to three other components: a bellows-operated vacuum pump that powers the entire mechanism, an intake hole that detects punched holes in the paper roll, and an third membrane connected to the hammer.

When the intake hole is blocked by the paper roll as in Figure 1.2B, the small volume between the intake hole and the membrane of the primary valve is depressurized as air is evacuated through a high-resistance bleed channel. With equal vacuum on both sides of the primary valve membrane, the valve remains in its resting state and vacuum is passed to the secondary valve membrane. The secondary valve also has equal vacuum on both sides of its membrane, so the valve remains in its resting state and vents the volume connected to the diaphragm that operates the hammer. With no vacuum inside this volume, the hammer remains at rest and no note is played.

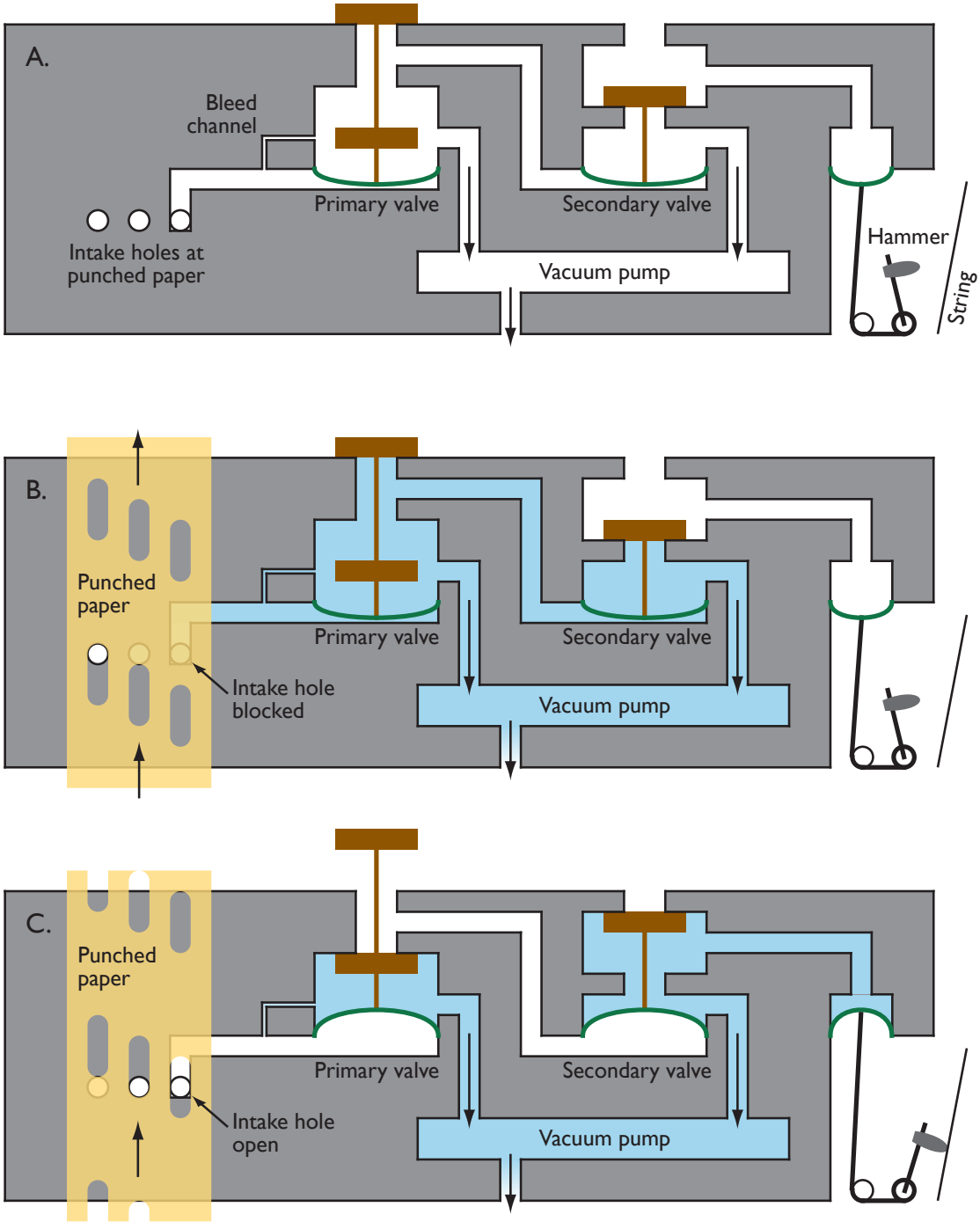


Figure 1.2: Schematic of a single key in the Pianola player piano at rest (A), with the intake hole blocked by the paper roll (B), and with the intake hole vented by a hole in the paper roll (C). Areas colored blue are under vacuum. After [4, 6, 7].

If a punched hole passes across the intake hole as in Figure 1.2C, the vacuum trapped between the intake hole and the primary valve membrane is vented through the intake hole. Note that even though the high-resistance bleed channel is still applying vacuum, the low-resistance open intake hole quickly vents any vacuum in the volume between the intake and the valve. The pressure differential across the primary valve membrane stretches the membrane and actuates the valve, thereby venting the volume that leads to the secondary valve membrane. The secondary valve then actuates, applying vacuum to the hammer diaphragm which in turn moves the hammer and plays the note [4, 6, 7].

What lessons about programmable pneumatic logic can be learned from the Pianola? First, *pneumatic valves can be used to actuate other pneumatic valves.* Valves can be arranged into circuits according to a few logical rules about pressure and vacuum applied to their inputs, outputs, and membranes. Second, *pneumatic resistance can be manipulated like electrical resistance.* A high-resistance bleed channel will depressurize a small *sealed* volume, but not if the volume is connected to the atmosphere by a low-resistance vent. Third, *efficient pneumatic amplifier circuits are feasible.* A small pressure change (the punched hole venting a small volume) is amplified into a large pressure change (the vacuum applied to the membrane that operates the hammer). Amplifiers are critically important in circuits where the output of one valve is fed as input to another valve. Fourth, *one section of the circuit can be vented without eliminating the vacuum in another section of the circuit.* Again, analogies to electri-

cal resistance can be applied. Fifth, *many pneumatic valves can be operated by few independent controllers through multiplexing.* Early pipe organs with multiple sets of pipes were controlled by a single keyboard using pneumatic multiplexing [4]. Sixth, *pneumatic actuation is scalable.* The Pianola contained *88* copies of the mechanism shown in Figure 1.2, arranged in parallel, actuated independently, and powered by a single vacuum pump. Finally, *simple and inexpensive punched-paper programs can be used to control complex pneumatic behavior.* Any desired actuation pattern (song or otherwise) can be programmed into punched paper and used to control a device. Applying these lessons to the control of microfluidic devices is a major goal of much of the research presented in this dissertation.[1]

1.3 von Neumann's automata

In 1948, the mathematician and physicist John von Neumann gave a talk at CalTech entitled "The general and logical theory of automata" [8]. Like Vaucanson, von Neumann was interested in the programmability of living organisms, but he also recognized that some aspects of living things could not be efficiently replicated by or programmed into the fledgling computers of the day. In particular, the fault-tolerance and evolutionary adaptability of living organisms had no analogue in electronic com-

[1] Technologies developed for player pianos played an unlikely role in the first successful aircraft simulator. In 1930, American player piano maker Edwin Link used player piano parts in his *Link Trainer.* Patented as both "an efficient aeronautical training aid [and] a novel, profitable entertainment device," the Link Trainer used punched paper programs to control pitch, roll, and yaw. It was used by Japan, the USSR, France, Britain, and Germany for fighter pilot training [4].

puters like the ENIAC. By recognizing that biomolecular systems had computational advantages over electronic ones, von Neumann set the stage for future DNA-based computations like the ones presented in this dissertation.

Von Neumann began his talk in 1948 with a comparison between the ENIAC and the human central nervous system. Though many times larger than the brain, the ENIAC had only 20,000 "switching organs" (vacuum tubes) compared to von Neumann's estimate of 10^{10} neurons in the brain. And while one of the ENIAC's vacuum tubes operated 1000 times faster than a neuron, the brain clearly possessed far greater computational powers than the ENIAC. Von Neumann blamed the poor performance of the ENIAC on the "inferiority of our materials, compared with those used in nature, which prevents us from attaining the high degree of complication and the small dimensions which have been attained by organisms" [8]. The use of biomolecular components gives "living computers" a fundamental advantage over their metal and glass counterparts. He continued,

> It is worth noting, although it is by no means surprising, how this divergence between objects, both of which are microscopic and are situated in the interior of the elementary components, leads to impressive macroscopic differences between organisms built upon them. This difference between a millimeter object and a micron object causes the ENIAC to weigh 30 tons and to dissipate 150 kilowatts of energy, while the human central nervous system, which is functionally about a million times larger, has the weight of the order of a pound and is accommodated within the human skull. [8]

For all their complexity, biological computers are also remarkably resilient compared to their manmade counterparts:

> Our present techniques involve the using of metals, with rather close spacings, and at certain critical points separated by vacuum only. This combination of media has a particular mechanical instability that is entirely alien to living nature. By this I mean the simple fact that, if a living organism is mechanically injured, it has a strong tendency to restore itself. If, on the other hand, we hit a man-made mechanism with a sledge hammer, no such restoring tendency is apparent. [8]

Von Neumann concluded that living organisms must "contain the necessary arrangements to diagnose errors as they occur, to readjust the organism so as to minimize the effects of the errors, and finally to correct or to block permanently the faulty components" [8]. In contrast, computers like the ENIAC were designed to make errors glaringly obvious so that the errors would be noticed and fixed by human operators.

So how could computers as powerful as the human brain be built? Researchers in the emerging field of neural networks suggested that the computational potential of the brain could be replicated using a very large neural network [9]. Von Neumann doubted the utility of this idea, fearing that any "neural network brain"

> . . . will turn out to be much larger than the one that we actually possess. It is possible that it will prove to be too large to fit into the physical universe. What then? Haven't we lost the true problem in the process? Thus the problem might better be viewed, not as one of imitating the functions of the central nervous system with just any kind of network, but rather as one of doing this with a network that will fit into the actual volume of the human brain. Or, better still, with one that can be kept going with our actual metabolistic "power supply" facilities, and that can be set up and organized by our actual genetic control facilities.

In other words, if a computer as small and powerful as the human brain is to be built, then *biomolecular components* should be used to build it, and *biochemical processes* like metabolism and molecular genetics should be used to control it.

Von Neumann concluded by offering one additional insight into how brain-like computers could be built. He noted that organisms *reproduce* and *grow more complex over many generations*, and he recognized that computers could be designed that make use of these behaviors. Von Neumann described a hypothetical automaton like Vaucanson's duck but capable of genetic variation and reproduction [8]. Each automaton contained three parts: an *instruction list* that provides a complete description of the automaton, a *supervisory unit* that oversees reproduction, and a *copying mechanism* that can take materials from the environment and build various things, including copies of the automaton itself [8, 10]. These artificial organisms could be programmed directly via the instruction list to perform a computation, but more importantly they could also *reproduce* and *evolve* to solve problems. By exerting environmental pressures on the reproducing automata (for example, allowing only the automata that correctly predict the folded structure of a protein to survive and reproduce), automata could be "bred" to solve a particular program:

> It is quite clear that the *instruction* . . . is roughly effecting the functions of a gene. It is also clear that the *copying mechanism* . . . performs the fundamental act of reproduction, the duplication of the genetic material, which is clearly the fundamental operation in the multiplication of living cells. It is also easy to see how arbitrary alterations of [the instruction] can exhibit certain typical traits which appear in connection with mutation, lethally as a rule, but with a possibility of continuing reproduction with a modification of traits [8].

This evolutionary process, von Neumann reasoned, could produce computers with capabilities approaching those of the human brain—after all, the evolutionary process created the human brain in the first place.

Incredibly, von Neumann's vision for evolutionary computing was presented in 1948, only two years after the ENIAC was unveiled and five years *before* Watson and Crick presented the structure of DNA. Although DNA was finally used to perform a computation nearly fifty years later [11], the use of DNA in an *evolutionary* computation remains a tantalizing but unrealized prospect nearly sixty years after von Neumann's talk. Progress toward realizing von Neumann's goal of evolutionary computing has been frustrated by Michael Conrad's *tradeoff principle*, as discussed in Section 1.5.

1.4 Cellular automata

While von Neumann's hypothetical reproducing automaton was never built, it did inspire another automaton that continues to play an influential role in the field of biomolecular computing. *Cellular automata* are discussed in two contexts in this thesis. First, advances in *DNA-based cellular automata* are reviewed later in this Introduction, and prospects for fabricating these nanostructures in microfluidic processors are discussed in Chapter 6. Second, actual *microfluidic cellular automata* are proposed in Chapter 6 as generic, programmable microfluidic processors that could be used for a variety of computational and analytical tasks. A brief introduction to cellular automata is presented here as a foundation for these later applications of cellular automata.

A cellular automaton can be thought of as several of von Neumann's automata

arranged in a grid or array. Each "cell" can communicate with neighboring cells, and each cell has a state or value that (in the simplest case) might be a single binary bit, TRUE or FALSE. Such a cellular automaton can be visualized as a grid of squares, with black squares representing TRUE cells and white squares representing FALSE cells. The black or white state of a cell is determined by the states of its neighboring cells. For example, in the most famous 2D cellular automaton, John Conway's *Game of Life*, a cell turns black or white depending on how many of the four neighboring cells are already black or white [12].

While the rules dictating the behavior of a cellular automaton can be very simple, the automaton's behavior can be very complex [13]. Four example cellular automata are presented in Figure 1.3. The automaton in Figure 1.3A follows a very simple rule—each cell is the opposite of the cell above it—and produces a simple repetitive pattern. The automata in Figure 1.3B follows a different rule: each cell is determined by the pattern of the *three* neighboring cells in the row directly above it. There are a total of 2^3 or 8 possible patterns for the neighboring row of cells, ranging from all white (□□□) to all black (■■■). In Figure 1.3B, a cell is black if its neighboring cells in the row above it are either ■■□, ■□□, □■■, or □□■; otherwise, the cell is white [13]. This simple rule produces the complex Sierpenski triangle fractal pattern shown in Figure 1.3B. A similar rule produces the extremely complex random pattern shown in Figure 1.3C. Finally, the automaton in Figure 1.3D represents the integers from 0 (top row) to 31 (bottom row) represented in binary (□ = 0 and ■ = 1) [14].

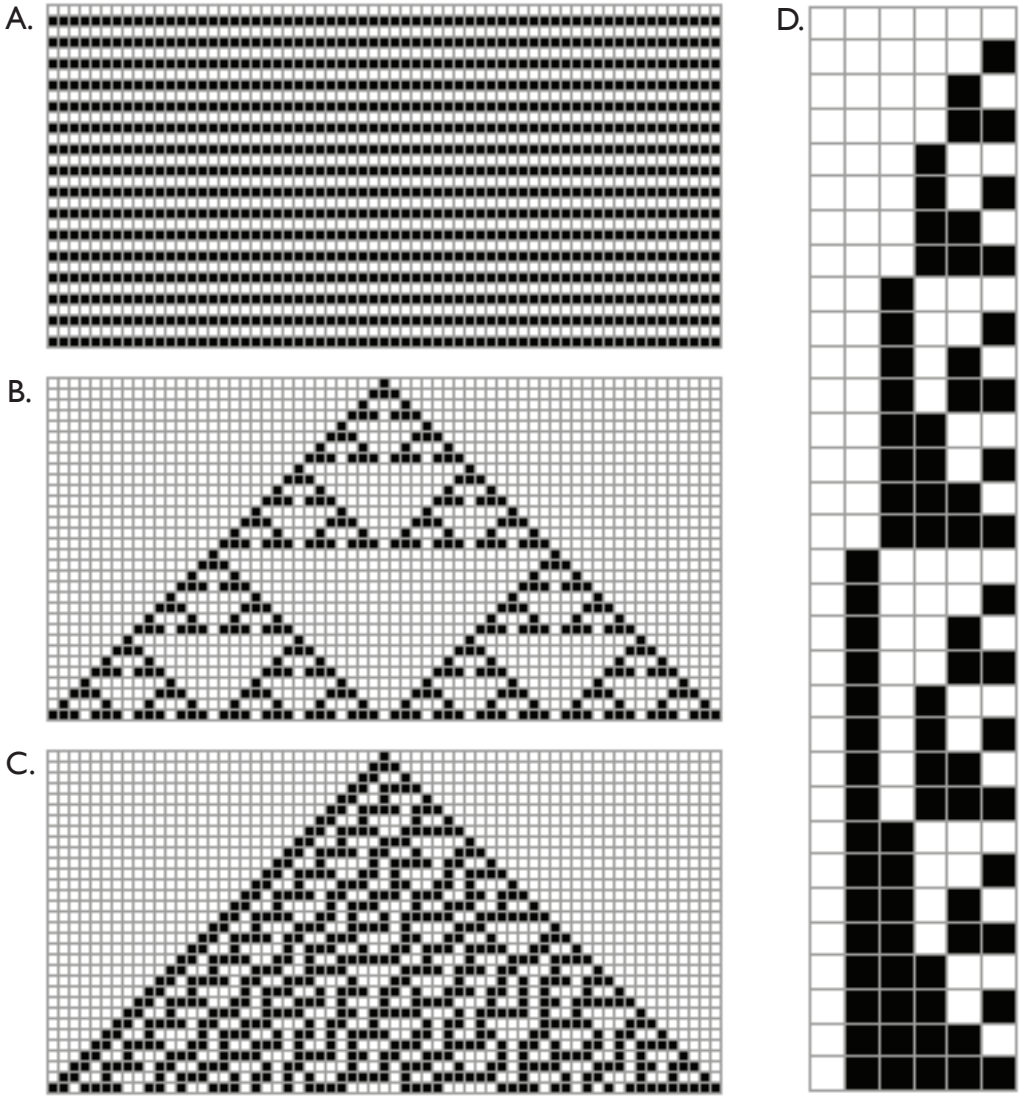


Figure 1.3: Cellular automata demonstrating repetitive (A), fractal (B, a Sierpenski triangle), random (C), and binary counting (D) behavior. After [13, 14].

What is the significance of cellular automata in molecular computing and microfluidic processor design? Since they operate according simple programs, *complex cellular automata can be constructed using simple materials*, including pieces of DNA or networks of microfluidic valves. As will be discussed in Section 1.6.3, every pattern shown in Figure 1.3 has been successfully constructed out of DNA. This field of DNA-based nanofabrication is one of the most active and promising outgrowths of DNA computing, and is discussed in detail later in this dissertation. Also, cellular automata serve as the inspiration for valve-based *microfluidic cellular automata* discussed later, devices that could be operated by simple programs and applied to a variety of complex computational and analytical problems.

1.5 Conrad's tradeoff principle

In the sixty years since von Neumann proposed that evolutionary biomolecular computers could be built that would approach the computing power of the human brain, no such computer has been built. On the contrary, modern computers are little more than faster versions of the ENIAC, and the flaws of the ENIAC—sensitivity to malfunctions, inefficiency, and a lack of evolutionary adaptability—still plague computers today. In 1990, Michael Conrad offered a rule to explain why von Neumann's vision had not been (and may never be) fully realized:

> A computing system cannot have all of the following three properties: *structural programmability*, *high computational efficiency*, and *high evolutionary adaptability*. Structural programmability and high computational

> efficiency are always mutually exclusive. Structural programmability and evolutionary adaptability are mutually exclusive. [15]

While this *tradeoff principle* imposes significant limits on both conventional and molecular computers, it also hints at the great potential for computers that occupy the middle-ground between programmability, evolvability, and efficiency—devices like the microfluidic processors presented in this dissertation.

Before his tradeoff principle can be discussed, Conrad's properties of computers must be defined. A computer has high *structural programmability* if all aspects of programming and operating the computer can be expressed a finite user manual [15]. This is a property of all electronic computers and their programming languages; any departure from the language's syntax is a bug, and only correctly-composed programs will compile and run. In contrast, evolving organisms cannot be programmed in a conventional sense. The molecular mechanism for evolution operates on a scale so small and massively parallel that, while well understood, cannot be directly programmed to operate in arbitrary ways. A true evolutionary computer could only be manipulated at a high level, by applying environmental stresses and letting the computer adapt and develop fitness over multiple generations. Conrad defines *computational efficiency* as the fraction of "all possible computational interactions" within a computer that are actually used while performing a computation [15]. A conventional computer uses only a miniscule fraction of the "all possible programs" that *could* be run on it, even though the set of "all possible programs" undoubtedly in-

cludes programs that are superior to the ones actually used on the computer. In contrast, evolving organisms survey the landscape of "all possible mutations" in the search for fitness. By operating in parallel and on a molecular scale, an evolutionary computer could operate with much greater computational efficiency than the most efficient traditional computer. Finally, a computer has high *evolutionary adaptability* if it employs evolutionary strategies to learn optimal solutions to problems [15]. As von Neumann observed, evolutionary adaptability is a property of all living organisms that is almost always absent from electronic computers [8]. Note that the term "evolutionary adaptability" applies not only to the mechanisms of Darwinian evolution but also to other molecular systems that operate with a high degree of parallelism and computational "fuzziness," like proteins folding. The fact that proteins fold without trying all possible folded conformations (the famous Levinthal Paradox [16]) is an example of evolutionary adaptability.

Why can a computer be structurally programmable or computationally efficient but never both? Increasing the programmability of a computer inevitably decreases the fraction of the "all possible programs" that could be used in that computer. For example, in a conventional computer, more programs are possible using a low-level programming tool like assembly language instead of a high-level language like FORTRAN or C. Reducing the number of "all possible programs" simplifies the task of the programmer but reduces the likelihood of finding an *efficient* program for a computer. Similarly, for an evolutionary computer, any change made to the molecular

machinery of evolution that would make the computer easier to program would also make it less efficient, since ultimately fewer mutations would be surveyed for fitness. In summary, a computer can either be *programmable but inefficient* (like modern computers) or *efficient but unprogrammable* (like evolutionary computers).

Why can a computer be structurally programmable or evolutionarily adaptable but never both?[2] Random mutations, which would play a critical role in the operation of an evolutionary computer, are almost always disastrous in structurally programmable computers. As von Neumann observed, rerouting a single wire in a conventional computer or changing a single binary bit in a conventional computer program will almost always "break" the computer or render the program unrunable [8]. In summary, a computer can either be *programmable but unadaptable* (like modern computers) or *adaptable but unprogrammable* (like evolutionary computers).

Although the tradeoff principle sets significant limits on the capabilities of both conventional and molecular computers, it also raises the prospect of *intermediate computers* that strike a balance between programmability, efficiency, and adaptability. These computers

> ... may operate in an intermediate zone, allowing for some instructive control on the part of a designer, but not precise prescriptive control. The potential efficiency would be greater than that for programmable digital machines... The difficulty of fabricating a purely organic version of a von Neumann machine at this time should not obscure the possibilities inherent in semi-organic approaches [15].

[2] A computer can demonstrate both structural programmability and evolutionary adaptability, but only at the expense of computational efficiency [15]. For example, a conventional computer can be programmed to run simulations of genetic or evolutionary algorithms, but these simulations invariably run much more slowly than their molecular-scale counterparts.

Such an intermediate computer would represent a compromise: while it would inevitably be less evolvable and less efficient than a purely-molecular computer, it would be more programmable (and hopefully more useful for solving real-world problems) than a purely-molecular computer. Conrad offered several examples of intermediate computers [15], including bacteriorhodopsin-based optical memories [17] and reaction waves propagating through Belousov-Zhabotinsky media ("clock reactions") in microreactors [18]. But these examples are still more programmable than evolvable: since all like molecules in a bacteriorhodopsin optical memory or Belousov-Zhabotinsky reaction are fundamentally identical and indistinguishable, these materials must be *compartmentalized* into pixels or chambers before they can be used in a digital computation. Even though they may contain millions of molecules, each pixel or microreactor functions as a single binary bit in a digital computation, and interactions between these pixels or chambers (not interactions between individual molecules) are used to perform a computation. The initial evolvability of the bacteriorhodopsin or Belousov-Zhabotinsky media is lost when it is partitioned into a programmable array of compartments, and the resulting structure offers few improvements over existing tools for data storage and computation.

A truly useful intermediate computer would retain more of the evolvable character of its molecular components, while imposing just enough programmability to make the computer practical for solving real-world problems. DNA is an ideal molecular material for such a computer. Digital information can be encoded into DNA on the molecular

scale in the base sequence, so partitioning DNA into bit-compartments is not necessary for DNA-based digital computing. Also, since genetic reactions (hybridization, extension, ligation, and so on) occur at the molecular level, computations using these massively-parallel molecular scale operations can have extremely high evolvability and efficiency. But while many approaches to DNA computing have been proposed and demonstrated (as discussed in the next section), the inherent unprogrammability of "bulk" DNA has limited its usefulness for solving many common problems. As a result, most DNA computations have been performed in test tubes containing sub-populations of DNA that can be operated upon (split, combined, etc.) in crudely programmable but labor-intensive steps [11]. A microfluidic DNA computer would be a more-programmable alternative to these test tube DNA computers. Such a device could operate upon hundreds or thousands of separate populations of DNA in parallel according to a program that encodes the order of operations required for the computation. This is the theoretical potential of the microfluidic DNA computers presented in this dissertation: to function as intermediate computers that combine aspects of programmability, evolvability, and efficiency to tackle real-world computational and analytical problems.

1.6 Foundations of DNA computing

In the rest of this Introduction, selected DNA-based computations that influenced the work presented in this thesis are reviewed.[3] A discussion of Adleman's seminal computation is followed by an analysis of the "all possible answers" method of DNA computing. Two more recent developments, computing by nanostructure self-assembly and evolutionary computing, are then presented. Finally, early attempts at automating DNA computation using microfluidics are discussed.

1.6.1 Adleman's computation

In 1994, Len Adleman used DNA to solve a seven-city instance of the traveling salesman problem [11]. Already legendary in computer science circles for co-discovering the RSA encryption algorithm [21], Adleman was quickly lauded as the father of the field of DNA computing as well, even though von Neumann had recognized the computational power of DNA over fifty years earlier. Regardless, Adleman was the first DNA computing *experimentalist*, and his primitive computation has had a profound influence on subsequent schemes for computing with DNA.

The problem solved by Adleman, the traveling salesman or Hamiltonian path problem, is a classic member of the *NP-complete* class of problems. The amount of computing resources (time or processors) required to solve these problems seems to

[3] This review focuses on *experimental* demonstrations of DNA computing. Several reviews of the *theory* of DNA computing are already available, e.g. [19, 20].

always scale *exponentially* as a function of the size of the problem [22]. In the worst case, *all possible answers* to a problem may have to be checked before a correct answer is found. In practice, well-designed algorithms can speed up the computation considerably, but solving a NP-complete problem in polynomial time (or better) appears to be impossible [22].

The instance of the traveling salesman problem solved by Adleman is shown in Figure 1.4. The problem asks, "given the cities and the allowed paths between the

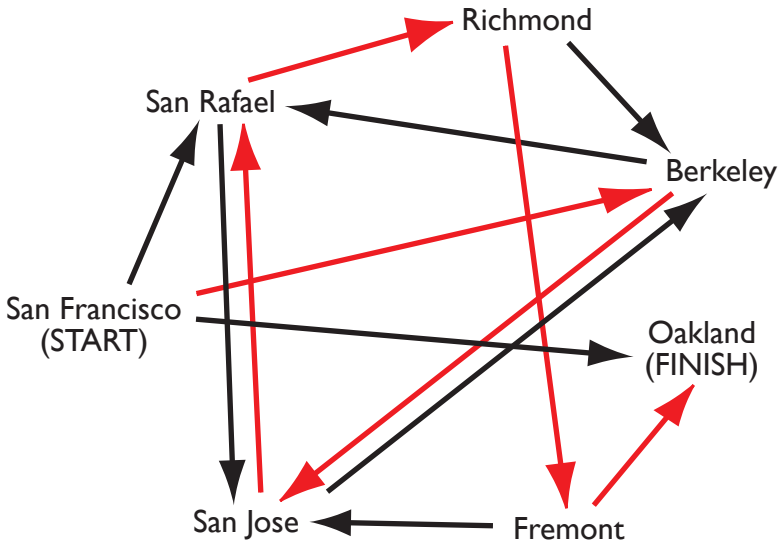


Figure 1.4: Adleman's traveling salesman problem (personalized for the Bay Area by the author). Arrows show allowed routes of travel between cities, and red arrows highlight the single satisfactory path: San Francisco → Berkeley → San Jose → San Rafael → Richmond → Fremont → Oakland. After [23].

cities shown in Figure 1.4, is there a path that starts in San Francisco, visits every city only once, and ends in Oakland?" Adleman solved the problem using a five-step algorithm:

1. Generate a population of input DNA molecules that represent random paths through the cities.

2. Keep only DNA representing paths that begin in San Francisco and end in Oakland.

3. Keep only DNA representing paths that enter a total of seven cities.

4. Keep only DNA representing paths that enter each city at least once.

5. If any input DNA remains, then the answer to the problem is *yes*. Otherwise, the answer is *no* [11].

To perform Step 1, Adleman first defined and synthesized seven 20-base DNA sequences and assigned one sequence to each city. Adleman also synthesized 20-base "splint" sequences for each of the 14 allowed routes of travel between cities; these sequences were complementary to both the last 10 bases of the "departure" city and the first 10 bases of the "arrival" city. When combined and ligated, these "city" and "splint" oligonucleotides self-assembled by hybridization to form the population of DNA representing random paths through the cities.

In Step 2, the input DNA population representing random routes was PCR-amplified using primers complementary to the "start" and "finish" cities. Only DNA representing paths that start in San Francisco and end in Oakland was amplified, and DNA representing paths that start or end in other cities was left at concentrations so low that they would not interfere with subsequent steps. In Step 3, the PCR product

from Step 2 was size-separated via agarose gel electrophoresis. The 140-base-pair band, which contained DNA representing paths through exactly seven cities, was excised and retained.

In Step 4, the remaining input DNA was rendered single-stranded and subjected to a series of seven sequence-specific "capture and release" steps. In each step, the input DNA was incubated with magnetic beads conjugated with 20-base oligonucleotides complementary to one of the seven cities. Input DNA complementary to the bead-immobilized oligonucleotide was captured, thermally released, and passed on to subsequent steps that selected for each of the remaining cities in turn. Any input DNA remaining at the end of the series of capture/release steps had to represent a path that visited each of the seven cities.

At this point in the computation, only correct answers (input DNA representing satisfactory paths through the cities) should remain. To detect the remaining DNA in the Step 5 and verify that it represents the correct answer to the problem, six additional PCR reactions were performed, each with reverse primers complimentary to a given city. The resulting amplicons were then size-separated by agarose gel electrophoresis, and the resulting series of fragment sizes was used to reconstruct the sequence of cities in the single (correct) remaining input DNA sequence.

Adleman's computation was undeniably labor-intensive. The solution of a seven-city traveling salesman problem required the synthesis of 21 input oligonucleotides and the performance of seven PCR amplifications, seven bead-based capture and re-

lease steps, and two gel electrophoresis separations. The entire computation required seven days of laboratory bench work [11]. Interestingly, Adleman notes that "Step 4 (magnetic bead separation) was the most labor-intensive, requiring a full day at the bench." These serial capture and release steps are particularly well-suited for automation on a microfluidic platform, as will be discussed in section 1.7 and Chapter 3.

1.6.2 "All possible answers:" promises and pitfalls

Adleman's computation [11] marked the first of many DNA computing schemes that relied upon encoding "all possible answers" to a problem in a population of DNA molecules as a first step in solving a problem. In subsequent steps, a series of separation operations retains potentially-correct DNA molecules and eliminates incorrect ones, and any DNA remaining at the end of the computation encodes the correct answer to the problem. In 1995, Richard Lipton proved that the "all possible answers" approach to DNA computing could be used to solve not just the traveling salesman problem but any NP-complete problem [24, 25]. In a very short time, the "all possible answers" method became a cornerstone of DNA computing.

The potential of the "all possible answers" method is striking. DNA is extremely well suited for storing and operating upon massive amounts of information in a miniscule volume, with a theoretical maximum of around one binary bit per nm^3 compared to 1 bit per 10^{12} nm^3 for existing magnetic storage media [11]. And while its individual

hybridization or ligation operations may be relatively slow, Adleman's "computer" performed approximately 10^{14} such operations in a few minutes. Finally, the energy requirements of operations like ligation ($\Delta G = 8$ kcal/mol or 5×10^{-20} joules per operation) approach the theoretical minimum of 2.9×10^{-21} joules per irreversible operation at room temperature [11]. DNA computers employing the "all possible answers" method could therefore have greater information storage density, faster processing speed, and better energy efficiency than practically any existing computer. Adleman concluded that, "for certain intrinsically complex problems... where electronic computers are very inefficient and where massively parallel searches can be organized to take advantage of the operations that molecular biology currently provides, it is conceivable that molecular computation might compete with electronic computation in the near term" [11].

Despite this extraordinary promise, Adleman and others also recognized many weaknesses in the "all possible answers" method of DNA computing. Adleman identified mismatches during hybridization as a critical source of errors in his computation [11,23]. Following ligation in Step 1, these "pseudopaths" that do not correspond to an allowed path could continue through the remaining computational steps and invalidate the readout in Step 5. Hybridization mismatches during the bead-based capture and release operations (Step 4) could have the same disastrous outcome. Adleman quantified these hybridization-based errors as ϵ_+ (the probability that DNA that should be captured from the population was accidentally left in the population

and discarded, constituting a *false negative*), and ϵ_-, the probability that DNA that should be left in the population was accidentally captured, constituting a *false positive*) [23]. Adleman estimated that the probability of a false positive during his computation was $\epsilon_- = 0.1$, and the probability of a false negative was $\epsilon_+ = 1 \times 10^{-6}$ [23]. Using experimental conditions similar to Adleman's, follow-up studies found ϵ_+ to be much higher than initially estimated: only 3-28% of perfectly-complementary 20-base DNA was captured by hybridization [26,27], and the rest was lost as "false negatives." Finally, Adleman was criticized for not including positive or negative controls in his computation, and efforts to repeat his experiment were deemed "inconclusive" [26].

Others have questioned whether DNA computers using "all possible answers" realistically have any chance of outperforming efficient algorithms on conventional computers. "Brute force" algorithms (including the "all possible answers" approach) rarely outperform well-designed algorithms algorithms on silicon computers [28]. The most complex DNA computation performed to date used the "all possible answers" method to solve a 20-bit, 24-clause instance of the satisfiability problem [29], and estimates suggest that traveling salesman problems larger than 70 cities cannot be solved using DNA [25]. In contrast, efficient algorithms can quickly solve much larger problems (1000 bits or more) on silicon computers [30]. A proposal to use "all possible answers" DNA computing to break a 56-bit DES (Data Encryption Standard) key [31] was expected to require four months of laboratory work [32] and would consume 10^{16} unique and *individually synthesized* oligonucleotides [33] to crack a code that can be

broken using current distributed computing techniques in less than 24 hours [34].

In spite of these limitations, the "all possible answers" model of DNA computing has served an important and influential role in the development of more-promising methods of DNA computing. Proposals for automating and miniaturizing "all possible answers"-based computations included the first applications of microfluidic technologies to DNA computing problems, as discussed in Section 1.7. By substituting clinical samples for "all possible answers" in the first step of a "computation," genetic assays can be performed—a prospect that is discussed in Chapter 6. Finally, by replacing the generation of all possible answers with an iterative *refinement* of existing answers in DNA or RNA computing, computations that are more biologically-inspired (and potentially more useful) than any other model of DNA computation could be performed. These evolutionary computation models are discussed in detail in Chapter 6.

1.6.3 DNA nanostructures: computing by self-assembly

In 1991, George Whitesides described "molecular self-assembly" as "the spontaneous association of molecules under equilibrium conditions into stable, structurally well-defined aggregates joined by noncovalent bonds" [35]. Whitesides recognized the ubiquity of this process in biological systems: tRNA folding into functional secondary structures, polypeptide chains folding into functional proteins, proteins assembling into functional aggregates, and so on. These self-assembled nanostructures could

serve as possible nanometer-scale "Erector sets" for a variety of nanotechnological applications.

Four years later, Erik Winfree proposed that specially-designed populations of DNA oligonucleotides could self-assemble into two- or three-dimensional networks, performing computations in the process [36]. Inspired by four-strand Holliday junctions that form during genetic recombination [37], Fu and Seeman had already demonstrated the feasibility of sequence-directed DNA-based nanostructures [38]. Winfree recognized the similarity between these lattice-like nanostructures and the cellular automata discussed in Section 1.3. Winfree postulated that carefully-designed DNA oligonucleotides could self-assemble into nanometer-scale tiles with dangling single-stranded oligonucleotide vertices. These blocks could then further self-assemble into large two- or three-dimensional networks, with the resulting structure determined by the complementarity between overlapping oligonucleotides on the individual tiles. Figure 1.5 depicts the fundamental three- and four-strand junctions used in DNA nanostructures, and a self-assembling lattice consisting of six-strand tiles with overlapping or "sticky" single-stranded vertices. By designing tiles with sticky DNA vertices that allow for only certain arrangements of the tiles, the self-assembly of the tiles can be controlled like a cellular automaton, and the solution to a computation can be represented in the resulting pattern of tiles [36].

Experimental results quickly verified the utility of Winfree's self assembly model of DNA computing. While agarose gel electrophoresis was initially used to roughly

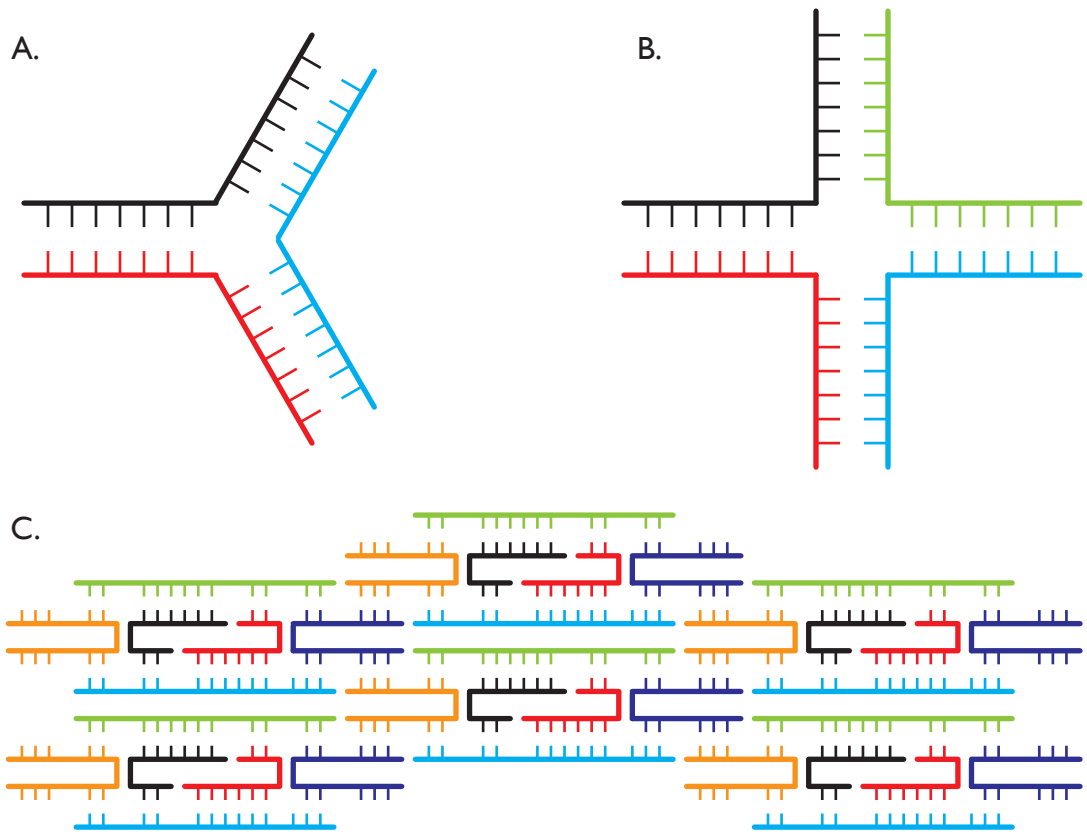


Figure 1.5: Three- and four-strand DNA junctions (A and B), and a self-assembled DNA lattice consisting of a network of smaller DNA tiles (C). After [36].

characterize the sizes and shapes of self-assembled DNA nanostructures [39], later studies used atomic force microscopy (AFM) to image the nanostructures and verify the presence of the expected pattern [40]. Different tiles are differentiated in AFM images by labeling the tiles with differently-sized DNA hairpins or nanocrystals. This method has been used to produce nanometer- to micron-sized DNA versions of all the cellular automata shown previously in Figure 1.3, include an alternating-stripes repetitive automaton [40], a fractal Sierpinski triangle automaton [41], and a binary counting automaton [14,42]. Not limited to cellular automata, the DNA self-assembly model has also been used to construct 100-nanometer squares, rectangles, triangles, stars, maps, "happy faces," and other arbitrary shapes [43].

Existing demonstrations of nanostructure-based DNA computing suffer from significant error rates, with 1 to 10% of tiles incorporated into incorrect positions [41]. Since an erroneous tile can influence the addition of neighboring tiles, the misplacement of a single tile often leads to cataclysmic errors in the final nanostructure [39]. These demonstrations typically use small, carefully-designed oligonucleotides to form the individual tiles and, in turn, the self-assembled tile nanostructure. Since the incorporation of a new tile is regulated by only a few hybridization events, *single* unintentional hybridizations can ruin an entire nanostructure. Also, all existing schemes of DNA computing by molecular self-assembly rely on atomic force microscopy for verification of expected nanostructures and readout of correct answers. While it provides unparalleled direct insight into the morphology of the nanostructures, AFM is a

notoriously slow and laborious process. AFM cannot realistically be used to analyze more than a few of the millions of nanostructures produced in a single reaction. In his first work on the subject, Winfree described an interesting alternative readout method: a special "reporter" oligonucleotide could be designed for incorporation into the nanostructure at the end of a series of expected tile assemblies. A sequence-specific enzyme could then bind to the reporter DNA and catalyze a phosphorescent reaction, resulting in a color change in the bulk solution [36]. While perhaps overly-complex, this readout scheme does provide a convenient binary "yes" (blue) or "no" (clear) readout without directly imaging the nanostructures by AFM. Microfluidic tools for reducing error rates in DNA nanostructures and reading-out the results of nanostructure-based computations are discussed later in the Prospects section.

1.7 Toward microfluidic DNA computing

The recognition of the need to *automate* DNA computations is a common theme across practically all paradigms of DNA computing. Automation is expected to improve the feasibility of DNA-based computations for two reasons. First, *automation will mechanize repetitive manual labor involved in computations.* Adleman's seminal computation, simple as it was, still required *seven days* of manual bench work [11]. In a sense, most existing models for DNA-based computation are not "DNA computers" but rather "human computers," people manipulating test tubes of DNA as one might manipulate the beads on an abacus. Without a human to operate them, both

the test tubes of DNA and the abacus are equally inept at performing computations. Automation thus becomes more than a simple labor-saving tool: it is an essential part of any proposal for a true DNA computer. Second, *automation will reduce or eliminate errors that have plagued manual DNA computations.* The proposed methods of error reduction discussed in this section would be difficult or impossible to implement without automated DNA computing hardware.

1.7.1 Theory of microfluidic DNA computing

Two theoretical descriptions of automated DNA computing are noteworthy because they anticipate the development of microfluidic DNA computers. In the first, John-Thones Amenyo observed in 1996 that "familiar computer design principles for electronic computers can be exploited to build practical computers at the mesoscopic scales of macromolecules and bio-polymers" [44]. Amenyo modeled the flow of DNA from step to step in a computation after the transmission of signals from point to point in electronic circuits. He advocated discarding the "test tube" model of DNA computing in favor of a microprocessor model: splitters, combiners, multiplexers and demultiplexers, and even transistor-like structures for operating on populations of DNA molecules.

In the second important theoretical work, Ashish Gehani described "micro flow bio-molecular computation," a conceptual microfluidic platform for performing DNA-based computations [45]. Gehani recognized the kinetic advantage of performing

reactions like hybridization at high concentrations in small volumes: "if we were able to place molecules between which an interaction was desired, into a smaller volume then they would interact much more quickly" [45]. He also observed that partitioning a reaction into several separate units was useful for avoiding errors caused by unintentional interactions between molecules. So a truly useful molecular computer would make provisions for both the combination of molecules when reactions are desired, and the separation of molecules when reactions are unwanted. As an example of such a system, Gehani offered living organisms: chemicals required for reactions in, say, neurons are produced or concentrated in these cells and kept out of other cell types where they could cause unintentional reactions.

Gehani then described a hypothetical molecular processor that would be suitable for both putting intended reactants together and keeping unintended reactants apart [45]. His processor consists of a three-dimension array of microfluidic chambers connected by valved channels. The valves are controlled by a (conventional) computer that also controls and receives feedback from actuators and sensors located in the chambers. DNA is routed from chamber to chamber like signals in a silicon microprocessor, as Amenyo had first suggested. Gehani's radical vision required hundreds of independently-controlled, densely-integrated valves and pumps—a level of fluidic large-scale integration that was not experimentally feasible in 1999.

1.7.2 Sticker computing and refinery models

Interest in automated DNA computing was catalyzed by the so-called "sticker" model of DNA computing originated by Roweis *et al.* in 1996 [46]. In "sticker" computations, short DNA oligonucleotides called "stickers" hybridize to specific locations on longer "memory" oligonucleotides. The presence of a "sticker" at a particular location on a "memory" oligonucleotide indicates that a particular bit on the "memory" oligonucleotide is set TRUE, and the absence of a "sticker" indicates that the bit is set FALSE. In contrast to the *all possible answers* method of computing, in which all assignments of TRUE and FALSE bits are set in fixed DNA sequences in the first step of the computation, bits encoded on "memory" oligonucleotides can be set TRUE or FALSE mid-computation by adding or removing "sticker" oligonucleotides.

Roweis designed "sticker" DNA computations to be performed automatically using a "parallel robotic workstation for molecular computation" [46]. His proposed system consisted of various flow-through tubes that could be connected in different arrangements to perform different operations. The selection of tubes included *data tubes* containing "memory" and "sticker" DNA, *separation tubes* containing immobilized capture oligonucleotides for sequence-specific capture of "memory" or "sticker" DNA, and *filter tubes* containing membranes that pass small "sticker" oligonucleotides but retain large "memory" DNA. To perform an operation like "extract all 'memory' strands for which Bit 1 is TRUE," the robotic workstation would connect the *data* tube containing the "memory" DNA to a *separation* tube with immobilized oligonu-

cleotides complimentary to the region of "memory" DNA rendered single-stranded if the associated "sticker" is missing (representing FALSE for Bit 1). A third tube would collect "memory" DNA that was not captured in the second tube because the Bit 1 sticker was present (representing TRUE for Bit 1). By connecting these tubes in series and circulating buffer through them using an external pump, the requested operation is performed and the correct "memory" DNA is collected in the third tube, ready for use in subsequent operations.

Perhaps the most provocative aspect of Roweis' robotic workstation was his proposed use of *refinery separations* to reduce the errors associated with operations like the one discussed above [46]. The refinery protocol was inspired by Adleman's earlier proposal to use *repetitive separations* as a strategy for reducing errors [23]. Consider a sequence-specific DNA separation operation with an acceptably low rate of false negatives (say, $\epsilon_+ = 0.01$) but an unacceptably high rate of false positives ($\epsilon_- = 0.1$). By collecting "positive" DNA (oligonucleotides containing the desired sequence) from one separation operation and subjecting these strands to a *second* separation identical to the first, the surviving "positive" DNA at the end of the second separation should contain errors at the lower rate $\epsilon_-^2 = (0.1)^2 = 0.01$. In general, n repeated separations should reduce the final error rate to ϵ^n.

Roweis expands upon the repetition model by replacing each sequence-specific separation step in his hypothetical robotic workstation with a "compound separator" modeled after fractionation columns used in oil refineries. A single separator and a

compound separator are compared in Figure 1.6. The compound separator consists

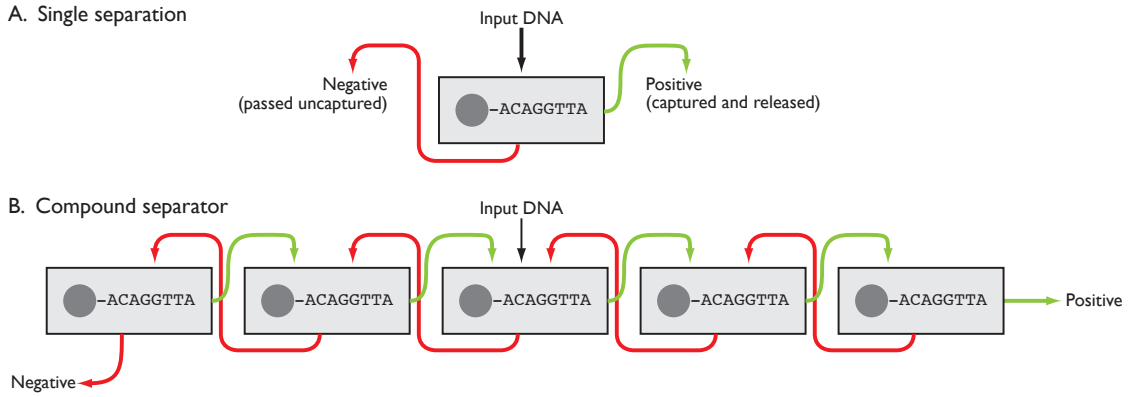


Figure 1.6: A schematic of a single separator (A) and a compound separator for improving error rates in automated DNA computations (B) while selecting for DNA complementary to the immobilized capture sequence. Green arrows show the preferred path of selected (perfectly-complementary) DNA and red arrows show the path of unselected (mismatched) DNA. After [46].

of several single separators arranged in parallel, so that selected (perfectly complementary) DNA from one separator is sent as input to the neighboring separator on the *right*, and unselected (mismatched) DNA is sent as input to the neighboring separator on the *left*. Input DNA that is passed out of the rightmost separator has a very high probability of being perfectly-complementary to the immobilized capture oligonucleotides in the separators, and input DNA that is passed out of the leftmost separator has a very high probability of *not* being perfectly-complementary to the capture oligonucleotides.[4] The compound separator is operated continuously; DNA passed from the extreme left and right separators is sent to additional compound sepa-

[4]Expressions for the error rates ϵ_+ and ϵ_- for an n-element compound separator are extremely complex due to the "random walk" aspects of the separation process, but are lower than the error rates associated with a system using n single separators arranged in series [46].

rators that select for additional sequences. This DNA "refinery" continues to operate until DNA representing the correct answer to a problem collects at an output.

Several practical considerations limited the ultimate feasibility of Roweis' robotic workstation and its compound separators when they were proposed in 1996. First, the robotic workstation was designed with filters that keep DNA from entering the pump and recirculating through the capture tubes [46]. This was done to avoid contaminating the pump with DNA, but the lack of recirculation limits the "memory" DNA to a single pass through the capture tubes. This inevitably reduces the capture efficiency of the workstation when compared to a system that could *recirculate* the "memory" strands through the capture tubes. Second, while the compound separator is an elegant solution to the ubiquitous problem of error reduction, realizing the compound separator in a parallel array of linked robotic workstations seems prohibitively complicated. Finally, the projected amount of DNA required to solve interesting problems on the robotic workstation seems wastefully large. Using the workstation to break a DES key (as described in section 1.6.2) would require a few thousand tubes, each containing a few *hundred milliliters* of DNA solution [32]. Despite these limitations, Roweis' robotic workstation proposal represents an important early step toward automated DNA computations, and possible microfluidic implementations of his compound separator are discussed in the Prospects section.

1.7.3 Gel-based DNA computing

Roweis' proposed robotic workstation inspired the first successful system for automated DNA computing, the gel-based DNA computer demonstrated by Braich *et al.* in 2001 [47]. Instead of beads, the gel-based computer used Acrydite-modified oligonucleotides in polyacrylamide gels for sequence-specific capture of DNA [48]. Gel-based capture more closely approximates solution-phase hybridization kinetics and is generally much more efficient that surface-based bead techniques. For this reason, even though input DNA was passed through the capture gel only once in the gel-based computer, the capture efficiency was still high enough to make the gel-based computer useful in large-scale DNA computations [47].

The first gel-based DNA computer utilized a 35 cm long glass tube containing a series of 11 capture gel plugs interspersed by plain agarose gel [47]. DNA representing all possible answers to a 6-bit Boolean satisfiability problem was then added to one end of the tube and driven electrokinetically to the first capture gel plug. Complementary DNA that satisfied the clause encoded in the first capture plug was captured by hybridization, and incorrect answers were carried by electrophoresis through the downstream capture plugs. Heaters were used to keep the *second* capture plug above the melting temperature of the DNA duplexes to avoid capturing the incorrect DNA passed through the first plug. After 30 minutes, the first capture plug was then heated to transfer the captured DNA to the second (now cooled) plug, and incorrect answers were passed electrokinetically through the *third* plug (heated to avoid capture). This

serial capture-and-release process was continued until only DNA representing the correct answer to the satisfiability problem remained in the last capture gel plug. 11% of the DNA representing the correct answer at the beginning of the computation was recovered at the end, indicating that the gel-based computer operated with ∼82% step-to-step transfer efficiency.

A similar gel-based DNA computer was used by Braich and colleagues to perform what remains the largest DNA computation yet, the solution of a 20-bit, 24-clause satisfiability problem [29]. This gel-based computer more closely resembled Roweis' robotic workstation: a *set* of glass tubes, each containing a single capture gel plug bracketed by agarose gel, replaced the single long glass tube used previously. During operation, two tubes were arranged end-to-end in a special gel electrophoresis box. The "origin" tube containing previously-captured DNA ready for release was maintained at a temperature above the melting point of the duplex, and the "destination" tube was kept below the melting point. The released DNA was then driven electrokinetically from the origin tube to the destination tube; satisfactory strands were captured by hybridization and incorrect strands were passed out of the tube and into the gel box. After four hours, the old "origin" tube was replaced by a new "destination" tube and the release-and-capture process was repeated. This continued through all 24 capture-and-release steps until only DNA representing the correct answer remained in the last tube. Between 61% and 87% of correct DNA was transferred successfully in each capture/release step [29].

Despite its extraordinary success (the 20-bit computation was *twice* the size of the largest DNA computation performed previously), the gel-based DNA computer represented an evolutionary dead end for the "all possible answers" class of DNA computers. Of the approximately 3×10^8 correct DNA molecules in the 500 pmol pool of all possible answers that went into the gel-based computer at the start of the computation, only about *5000* remained for readout at the end of the computation [29]. This very low yield is a result of low step-to-step transfer efficiencies during the computation: even though 87% transfer efficiency is high for DNA computing, losing 13% of the correct answer at each of 20 steps leaves very little correct DNA remaining at the end of the computation. Even if the transfer efficiency could be increased to nearly 100%, the challenge of discriminating individual bit sequences would continue to plague larger computations performed using the "all possible answers" approach. Using 15-base sequences to represent binary bits, *40* unique 15-mer sequences were used to construct the 300-base-long input DNA used in the 20-bit computation [29]. As more bit sequences are defined, the input DNA grows longer and more cumbersome to prepare, and the likelihood of unwanted interactions between partially-complementary sequences increases dramatically [49]. Braich admits that satisfiabilty problems larger than 30 bits would be extremely difficult to solve on the gel-based DNA computer without using mid-computation PCR amplifications to replenish dwindling numbers of molecules [29]. It is reasonable to conclude that "all possible answer" DNA computers will never compete with silicon computers opti-

mized to quickly solve 1000-bit problems [30]. Nevertheless, the "all possible answer" method remains a meaningful benchmark for new DNA computers (including the molecular processor presented in Section 2), and haplotyping assays based on the "all possible answers" method (discussed in Chapter 6) may prove useful for diagnosing complex genetic disease traits.

1.7.4 On-chip DNA capture and release

The first step toward an actual *microfluidic* DNA computer was made in 2001, when McCaskill described a on-chip system for serial capture and release of DNA [50]. Central to McCaskill's proposal were H-shaped channel junctions called "selective transfer modules" shown in Figure 1.7. Each module consists of two neighboring channels, one containing input DNA in hybridization buffer and the other containing an alkaline solution. Although the DNA and alkaline streams contact each other within the module, the laminar flow profile through the module limits cross-contamination between the streams. Magnetic beads derivatized with capture oligonucleotides are initially held in input DNA stream by an external magnet. As input DNA is pumped through the beads, perfectly-complementary (correct) DNA is captured by hybridization on the beads, and mismatched (incorrect) DNA is passed out of the module. The beads are then transferred magnetically into the alkaline stream, where the high pH destabilizes the DNA duplexes and releases the captured input DNA into the alkaline stream. Downstream mixing with fresh hybridization buffer neutralizes the alkaline

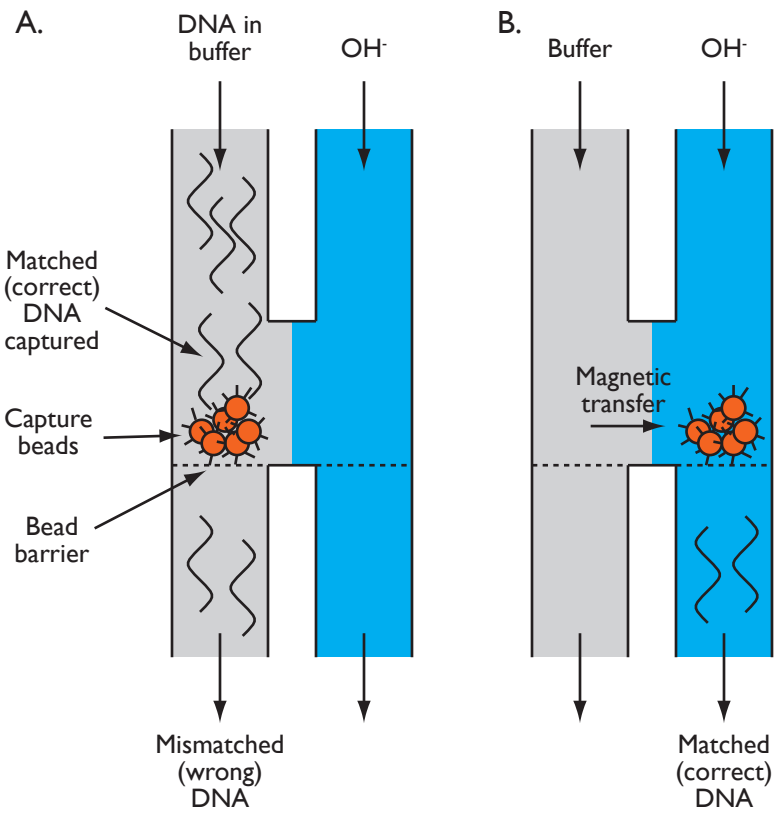


Figure 1.7: McCaskill's selective transfer module. After [50].

stream and prepares the released DNA for additional capture/release steps, and input DNA remaining after a series of capture/release steps represents the correct solution to the problem encoded in the series of capture/release steps [50].

Although McCaskill's proposed microfluidic computer had flaws that complicated its use in actual computations, certain aspects of the computer were demonstrated successfully and incorporated into subsequent proposals for microfluidic DNA computing. Magnetic bead manipulation via external magnetic fields was demonstrated and found to be a robust and scalable technique: many on-chip bead boluses could be held in place and shifted back and forth in parallel using arrays of off-chip magnets [51]. Later designs showed that bead rinsing steps (to reduce nonspecific capture of incorrect DNA) could be incorporated into the strand transfer modules [52]. Perhaps most importantly, McCaskill's proposal showed how the problem of interest could be programmed into a microfluidic device, in the order of capture/release steps and the path followed by input DNA through the device [53].

Unfortunately, the step-to-step DNA transfer efficiency of perfectly-complementary (correct) DNA between selective transfer modules was found to be around 50%, too low for the structures to be of much use in solving large-scale problems [54]. Two factors probably contributed to the low step-to-step transfer efficiency of the strand transfer modules. First, continuous-flow pH-mediated denaturation and neutralization is experimentally very difficult. The beads must be rinsed with a solution alkaline enough to release the captured DNA but weak enough to be neutralized downstream

with hybridization buffer without diluting the released DNA unnecessarily. Without specialized mixing structures like pumps or interdigitated channels, incomplete neutralization of the released DNA is a possibility. Since the formation of double-stranded DNA is only favorable in the narrow pH range between 5 and 9 [55], failure to neutralize the released DNA stream would greatly reduce the efficiency of the next capture step. Second, McCaskill's computer utilized only *a single pass* of the input DNA solution through the capture beads in the strand transfer modules. Without on-chip pumps to *recirculate* the input DNA through the capture beads, only a single pass through the capture beads is possible and the capture efficiency is correspondingly low. Recently-proposed devices attempted to solve these problems by using *negative selection* (wrong answers are captured, so pH-mediated release is unnecessary) and *recirculating soft-lithography loops* (to improve the capture efficiency) [56,57], but DNA computations performed using these proposed devices have not yet been presented.

Chapter 2

Monolithic Membrane Valves and Diaphragm Pumps for Practical Large-scale Integration into Glass Microfluidic Devices

Reproduced from "Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices" by William H. Grover, Alison M. Skelley, Chung N. Liu, Eric T. Lagally, and Richard A. Mathies, *Sensors and Actuators B* **89** (3), 2003 (315–323), with permission of Elsevier.

Monolithic elastomer membrane valves and diaphragm pumps suitable for large-scale integration into glass microfluidic analysis devices are fabricated and characterized. Valves and pumps are fabricated by sandwiching an elastomer membrane between etched glass fluidic channel and manifold wafers. A three-layer valve and pump design features simple non-thermal device bonding and a hybrid glass-PDMS fluidic channel; a four-layer structure includes a glass fluidic system with minimal fluid-elastomer contact for improved chemical and biochemical compatibility. The pneumatically actuated valves have <10 nl dead volumes, can be fabricated in dense arrays, and can be addressed in parallel via an integrated manifold. The membrane valves provide flow rates up to 380 nl/s at 30 kPa driving pressure and seal reliably against fluid pressures as high as 75 kPa. The diaphragm pumps are self-priming, pump from a few nanoliters to a few microliters per cycle at overall rates from 1 to over 100 nl/s, and can reliably pump against 42 kPa pressure heads. These valves and pumps provide a facile and reliable integrated technology for fluid manipulation in complex glass microfluidic and electrophoretic analysis devices.

2.1 Introduction

Microfluidic lab-on-a-chip analyzers have advanced rapidly from early single-channel devices [58] to current complex systems that can perform a wide variety of assays [59, 60]. Successful microfluidic assays have included polymorphism detection for breast cancer risk assessment [61], parallel combinatorial synthesis and analysis of chemical libraries [62], high-throughput genotyping [63] and DNA sequencing [64], chemical and biological antigen detection [65], and chiral resolution of amino acids for exobiological analysis [66]. However, the development of complete integrated systems for on-chip sample preparation and manipulation has shown more modest growth. Thus far, automated HIV genotyping has been demonstrated in a polymer microfluidic device that combines purification, amplification, and microarray hybridization steps [67], DNA amplification and integrated electrophoretic analysis in a glass microfluidic device has been demonstrated using individually addressed valves and vents to isolate fluids [68], automated protein sizing has been performed in a glass device using pressure-driven flow and electrophoresis to route fluids [69], and automated pathogen detection has been demonstrated in a micromachined polymer device utilizing membrane valves and pumps [70]. Complex fabrication, chemical compatibility, and unreliable fluid manipulation, among other problems, have made existing fluidic manipulation technologies disadvantageous for integration into large-scale, high-throughput lab-on-a-chip devices. A useful on-chip mechanism for nl- to μl-scale fluid manipulation must be compatible with the assay chemistry, be able

to accurately and reliably meter known volumes of fluid, and be amenable to facile large-scale integration.

A variety of microfabricated valves and pumps have been developed for on-chip fluidic manipulation and control. The earliest examples were fabricated using anodically bonded silicon and glass wafers and actuated piezoelectrically [71, 72]. The electrical conductivity and chemical compatibility of silicon, which can complicate its use in analytical devices, can be mitigated in part by the use of deposited chemically and electrically resistant thin films [73]. Flexible membranes can also be used to form the active elements in pneumatically actuated microfluidic valves and pumps. A variety of membrane-based valves and pumps have been demonstrated for silicon [74–76], glass-silicon [77, 78], and polymer [67, 79–81] microfluidic devices. In addition, the popularity of "soft lithography" [82] has led to the development of pneumatic valves and pumps suitable for integration into all-elastomer devices [83, 84]. While these demonstrations of elastomeric valves and pumps are encouraging, the hydrophobicity and porosity of many native elastomer surfaces render these valves and pumps incompatible with many chemical and biochemical assays unless surface modification chemistries are employed [85, 86]. Also, while successful fluorescence detection in an elastomer device has been demonstrated [87, 88], the native fluorescence of elastomeric material under visible light is significantly higher than that of glass.[1] This

[1] A sample of PDMS membrane used in this study was found to be over thirty times more fluorescent than an equal thickness of borosilicate glass. The samples were illuminated at 488 nm; emitted light from 535 to 565 nm was collected through a band pass filter and detected using a CCD camera.

presents a problem for our applications that demand high sensitivity detection. Finally, a variety of pumping methods based on electroosmotic flow (EOF) have been demonstrated [89–91]. These methods are useful for many applications although the sensitivity of EOF to analyte osmotic strength and surface contamination must be noted.

Glass microfluidic devices [58] have dominated applications where precise control of the fluidic channel surface chemistry, high quality electrophoretic separation, and high-sensitivity fluorescence detection are required. The variety of glass silanization chemistries [92] coupled with the insulating nature of glass make it particularly well-suited for use in capillary electrophoresis (CE) analysis devices [93]. The success of pneumatically actuated valves for polymer microfluidic devices [67] inspired the development of a pneumatic valve suitable for integration into glass analytical devices [94]. While this valve has been used successfully in a variety of glass CE devices [68, 95, 96], its reliance on individually placed latex membranes is problematic for large-scale integration into high-throughput devices.

Here we present the fabrication and characterization of membrane valves and diaphragm pumps that can be used for facile large-scale integration into glass microfluidic analysis devices. The valves and pumps employ a monolithic polydimethylsiloxane (PDMS) elastomer membrane, an integrated microfabricated manifold that provides independent addressing or parallel pneumatic actuation of arrays of valves

and pumps, and a glass fluidic system that minimizes fluid-elastomer contact.[2] The ease of fabrication and reliability of these valves and pumps will facilitate the development of high-throughput glass microfluidic devices.

2.2 Materials and methods

2.2.1 Microfabrication

The three- and four-layer device topologies used to fabricate monolithic membrane valves and pumps are illustrated in Figure 2.1 Channel features were etched into glass wafers using standard wet chemical etching [93, 95]. Glass wafers (1.1 mm thick, 100 mm diameter) were piranha cleaned (20:1 H_2SO_4:H_2O_2) and coated with a sacrificial 200 nm polysilicon layer using an LPCVD furnace or sputtering system. Borofloat glass wafers were used for devices with the three-layer design and D263 borosilicate glass wafers were used for devices with the four-layer design. After polysilicon deposition, the wafers were spin-coated with positive photoresist, soft-baked, and patterned using a contact aligner. UV-exposed regions of photoresist were removed in Microposit developer. The exposed regions of polysilicon were removed by etching in SF6 plasma. The wafers were etched isotropically at 7 μm/min in HF solution (49% HF for the Borofloat wafers and 1:1:2 HF:HCl:H_2O for the D263 wafers) until the desired etch depth was reached. The fluidic channel wafers were etched 20 μm deep for the

[2] A preliminary presentation of this work is found in [97].

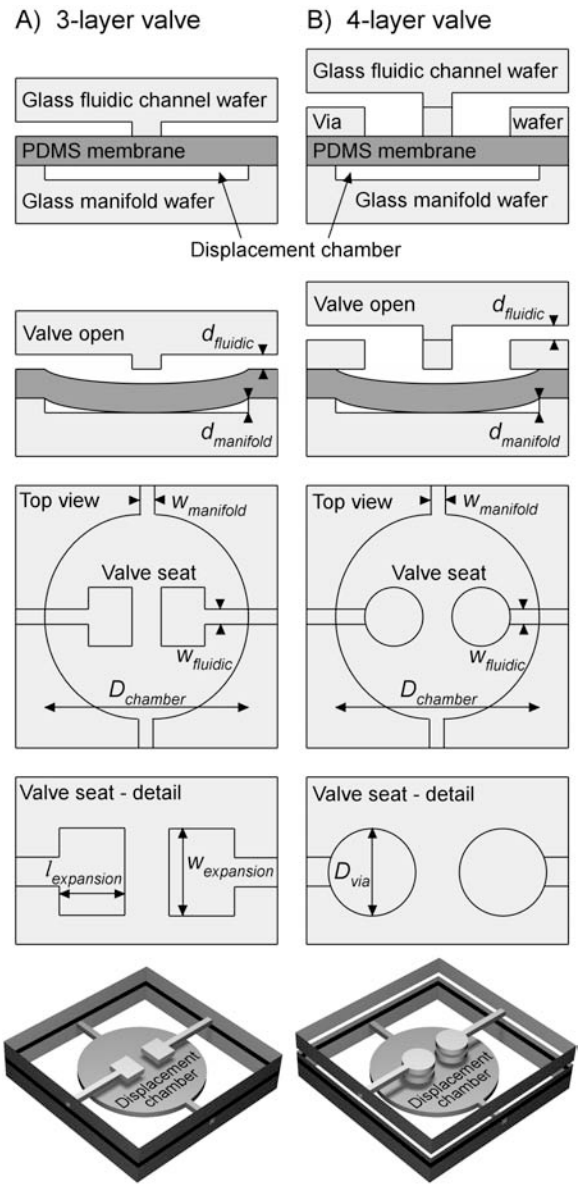


Figure 2.1: Cross-sectional, top, and oblique views of three-layer (A) and four-layer (B) monolithic PDMS membrane valves. Each valve consists of a glass manifold with an etched displacement chamber for pneumatic actuation, a working PDMS membrane, and a glass fluidic channel wafer containing the channel to be valved. In the three-layer topology, PDMS defines one surface of the valved channel. In the four-layer structure, the addition of the drilled via wafer defines all-glass fluidic channels with minimal fluid-PMDS contact. The three-layer valve includes a channel expansion in the valve seat with dimensions $w_{expansion}$ by $l_{expansion}$, while the four-layer valve includes drilled via holes with diameter D_{via}. The wafer etch depths are $d_{fluidic}$ and $d_{manifold}$, channel widths are $w_{fluidic}$ and $w_{manifold}$, and displacement chamber diameter is $D_{chamber}$.

three-layer devices and 40 μm deep for the four-layer devices. The manifold wafers were etched 70 μm deep for the three-layer devices and drilled at valve locations for the four-layer devices. The remaining photoresist and polysilicon was then stripped from the wafers using PRS-3000 and SF_6 plasma, respectively. Access holes through the fluidic and manifold wafers were drilled and the wafers were again piranha cleaned.

Devices utilizing the three-layer design shown in Figure 2.1A were assembled by applying a PDMS (polydimethylsiloxane) membrane (254 μm thick HT-6135 and HT-6240, Bisco Silicones, Elk Grove, IL) over the etched features in the fluidic channel wafer and pressing the manifold wafer onto the PDMS membrane. This process formed hybrid glass-PDMS fluidic channels with valves located wherever a drilled or etched displacement chamber on the manifold was oriented directly across the PDMS membrane from a valve seat. Devices utilizing the four-layer design in Figure 2.1B were assembled by first thermally bonding the fluidic channel wafer to a 210 μm thick D263 via wafer with pairs of 254 μm diameter drilled via holes positioned to correspond to the locations of channel gaps. The fluidic channel and via wafers were bonded by heating at 570 °C for 3.5 h in a vacuum furnace (J.M. Ney, Yucaipa, CA). The resulting two-layer structure containing all-glass channels was then bonded to the PDMS membrane and the manifold wafer as described above. The glass-PDMS bonds formed in this manner were reversible but still strong enough to survive the range of vacuum and pressures exerted on the device. Optionally, an irreversible glass-PDMS bond was obtained by cleaning the manifold wafer and PDMS membrane in a UV

ozone cleaner (Jelight Company Inc., Irvine, CA) immediately prior to assembly.

2.2.2 Operation and characterization

The monolithic membrane valves with integrated manifolds were actuated by vacuum or pressure applied to pneumatic connections on the device and distributed by the channels in the manifold wafer to the displacement chambers. Similarly, monolithic membrane valves without integrated manifolds were actuated by applying vacuum or pressure directly to drilled displacement chambers on the manifold wafer beneath each valve. Applying a vacuum deflected the PDMS membrane into the displacement chambers, thereby allowing fluid to flow across the gaps in the fluid channels. Applying pressure forced the PMDS membrane against the fluidic channel wafer, thereby stopping the flow of fluid. Valve actuation was found to occur in two steps, with only the regions of membrane directly below the fluid channels deflecting at first, then the remainder of the membrane separating from the valve seat on the fluidic channel wafer and deflecting into the displacement chamber as the valve opened fully. Expanding the end of the fluidic channel within the valve seat was found to decrease the pressure differential required to initiate the first step in valve actuation. For this reason, expanded fluidic channels were included in all valves used in this study. Pressure and vacuum for valve actuation were controlled by a set of solenoid valves (Humphrey Products, Kalamazoo, MI) and a computer running LabVIEW (National Instruments, Austin, TX); measurements of actuation pressure or

vacuum were relative to atmospheric.

Three valves placed in series form a diaphragm pump, as shown in Figure 2.2. The three-layer diaphragm pump test wafer shown in Figure 2.2 contains 144 valves configured to form 48 different pumps. Pumping was realized by actuating the input, diaphragm, and output valves of each pump according to the five-step cycle shown in Figure 2.3. The 48 pumps in the test device were actuated in parallel via three sets of three pneumatic connections (A, B, and C in Figure 2.2) on the underside (manifold wafer) of the device.

The dependence of pump performance on diaphragm valve displacement chamber volume was characterized using pumps 1 through 12. The pre-etch diaphragm valve displacement chamber diameter $D_{chamber}$ and post-etch chamber volume $V_{chamber}$ for pumps 1 through 12 are summarized in Table 2.1. $V_{chamber}$ was calculated using an isotropic etch model

$$V_{chamber} = \frac{1}{4}\pi D_{chamber}^2 d_{manifold} + \frac{1}{4}\pi^2 D_{chamber} d_{manifold}^2 \tag{2.1}$$

The identical input and output valves utilized 1.6 mm $\times$ 1.8 mm hexagonal displacement chambers with chamber volume $V_{chamber} = 270$ nl. The input, diaphragm, and output valves in pumps 1 through 12 had identical fluidic channel expansions ($w_{expansion} = 300$ μm, $l_{expansion} = 500$ μm) separated by a small 500 μm gap.

The dependence of pump performance on fluidic channel cross-sectional area was characterized using pumps 25, 28, 30, 33, 35, and 38. The pre-etch fluidic channel width $w_{fluidic}$ and post-etch channel cross-sectional area $A_{fluidic}$ for the six pumps are

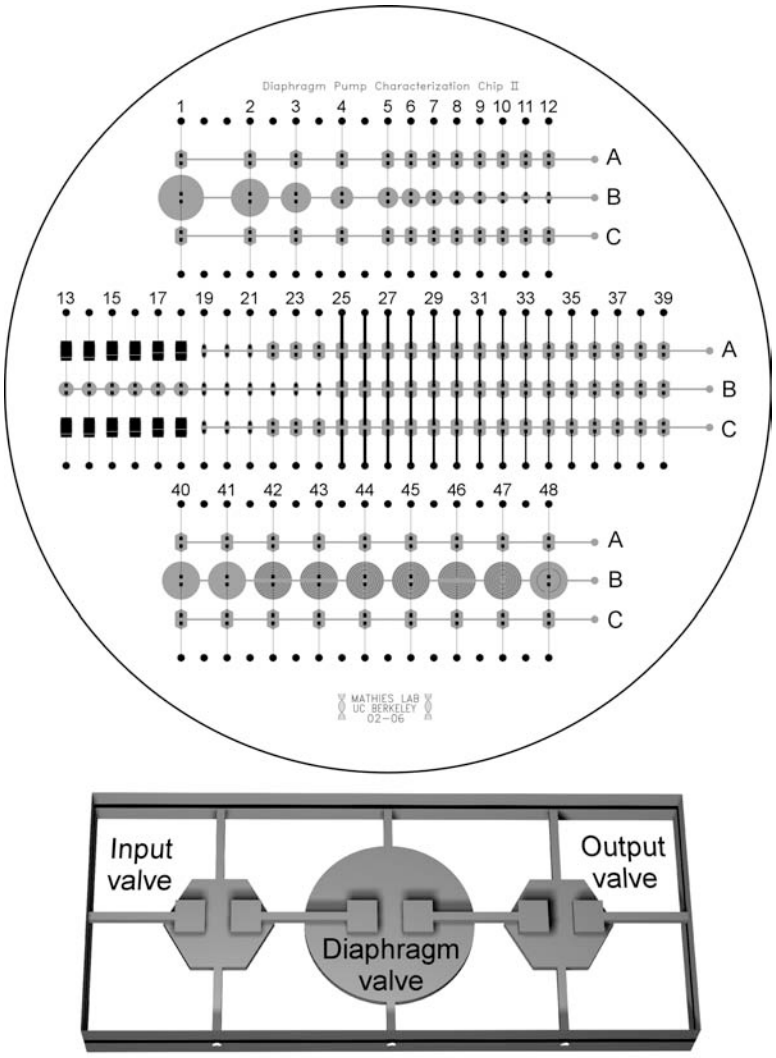


Figure 2.2: Three-layer diaphragm pump characterization wafer. The wafer contains 144 valves arranged to form 48 different pumps; critical dimensions of the pumps are summarized in Table 2.1. Pneumatic connections at drilled holes A, B, and C are used to actuate each series of pumps in parallel. Pumps 40 through 48 were designed to test different valve seat fluidic channel geometries. Pumps with meandering or interdigitated fluidic channels in the valve seats (pumps 42 through 48) were found to be more resistant to bubble entrapment and pump more reliably than pumps with standard valve seats (pumps 40 and 41). Inset shows an oblique view of one pump.

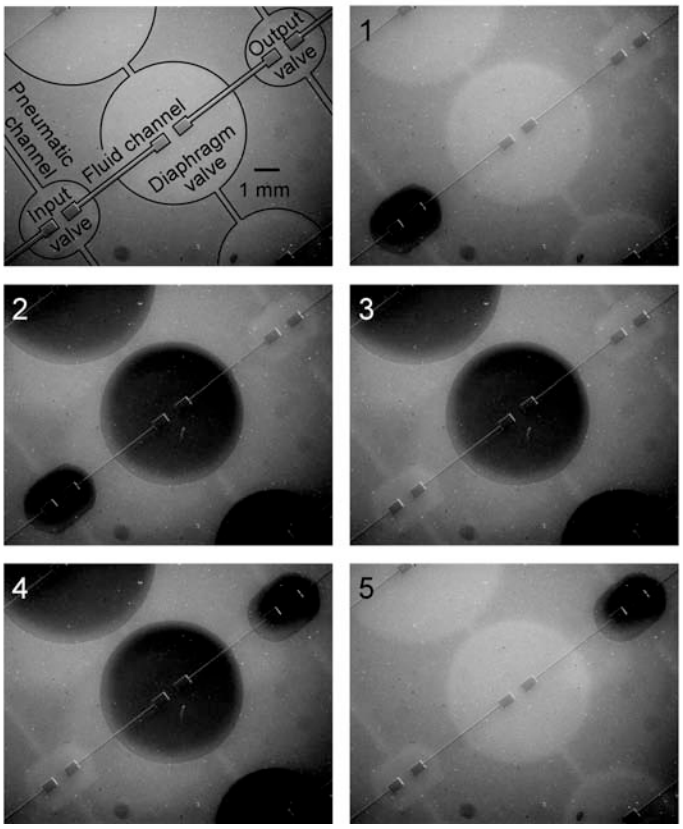


Figure 2.3: Darkfield images showing the five steps in the diaphragm pumping cycle: (1) open input valve and close output valve, (2) open diaphragm valve, (3) close input valve, thereby defining the volume pumped per cycle as the volume contained within the open diaphragm valve, (4) open output valve, (5) close diaphragm valve.

Pump	$D_{chamber}$ (μm)	$V_{chamber}$ (nL)
1	6000	2050
2	5000	1430
3	4000	928
4	3000	531
5	2750	449
6	2500	374
7	2250	306
8	2000	244
9	1750	198
10	1500	142
11	1250	101
12	1000	67.1
	$w_{fluidic}$ (μm)	$A_{fluidic}$ (μm^2)
25	300	6630
28	240	5430
30	200	4630
33	140	3430
35	100	2630
38	40	1430

Table 2.1: Diaphragm pump dimensions

reported in Table 1. $A_{fluidic}$ was calculated using an isotropic etch model

$$A_{fluidic} = w_{fluidic} d_{fluidic} + \frac{1}{2}\pi d_{fluidic}^2 \tag{2.2}$$

Pumps 25, 28, 30, 33, 35, and 38 had identical input, output, and diaphragm valves consisting of 1.6 mm × 1.8 mm hexagonal displacement chambers with chamber volume $V_{chamber}$ = 270 nl, $w_{expansion}$ = 300 μm, $l_{expansion}$ = 500 μm separated by a 500 μm gap.

2.3 Results

2.3.1 Valve characterization

Figure 2.4A presents a characterization of water flow through a four-layer monolithic membrane valve. The valve offered very little resistance to fluid flow at manifold pressures below 0 kPa. Increasing the manifold pressure at the PDMS membrane quickly increased the amount of pressure required to break fluid through the channel gap. Applying a manifold pressure of only 10 kPa effectively sealed the valve against 40 kPa fluid pressure, and a manifold pressure of 45 kPa sealed the valve against fluid pressures as high as 75 kPa. Acting over a membrane surface area of 50,000 μm^2, the 75 kPa fluid pressure exerted a force of 3.8 mN on the membrane. The manifold pressure acted over a much larger section of membrane (1.8 mm^2) but the flexible membrane applied most of this force to the wafer and only a fraction of the net manifold force actively counteracted the fluid force. Still, the valve sealed successfully

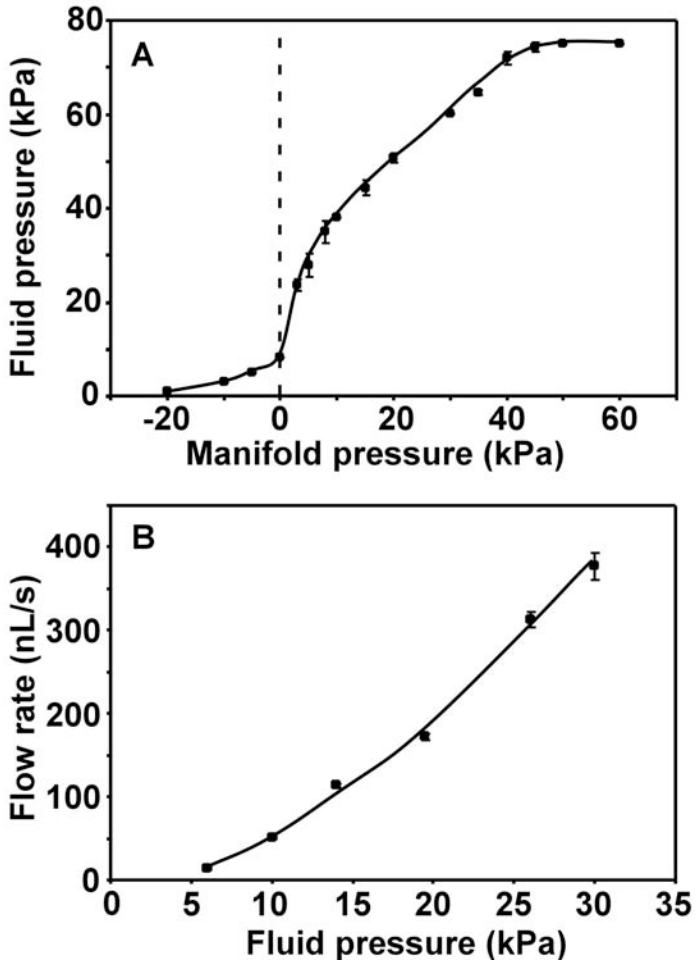


Figure 2.4: (A) Fluid pressure required to initiate water flow through a valve being held at the indicated manifold pressure. (B) Flow rate of water through a valve as a function of pressure applied to the fluid while holding the valve open with a constant vacuum of −30 kPa. A four-layer valve with a HT-6240 PDMS membrane was used. Valve dimensions were $d_{fluidic} = 40$ μm, $d_{manifold} = 1100$ μm, $w_{fluidic} = 100$ μm, $D_{via} = 254$ μm, and $D_{chamber} = 1500$ μm. The calculated dead volume of the valve (based on the volume of the two drilled via holes) was 20 nl; three-layer monolithic membrane valves with similar dimensions had 8 nl calculated dead volumes.

against a fluid pressure nearly double the manifold pressure, and valves with thicker or less elastic membranes would be expected to hold off even greater fluid pressures at the same manifold pressure.

Figure 2.4B presents the rate of water flow through an open monolithic membrane valve as a function of applied fluid pressure. The rate of fluid flow through the valve was found to have a roughly linear dependence upon the fluid pressure, and flow rates as high as 380 nl/s were attainable for the valve tested. The initial resistance to flow between 0 and 5 kPa fluid pressure was attributed to the hydrophobic nature of the PDMS membrane. For this and other three- and four-layer valves with fluidic channel cross-sectional areas ($A_{fluidic}$) much smaller than the cross-sectional area of the fluid path through the open valve, the overall rate of fluid flow is primarily determined by $A_{fluidic}$.

2.3.2 Diaphragm pump characterization

The 48-pump device shown in Figure 2.2 was used to characterize the performance of diaphragm pumps constructed from monolithic membrane valves. Figure 2.5 plots the maximum volume pumped per cycle versus the displacement chamber volume $V_{chamber}$ of the diaphragm (central) valve for pumps 1 through 12 at zero pressure head. All five steps in the pumping cycle were given excess time to occur (1.5 s for steps 1, 3, and 4 and up to 10 s for steps 2 and 5) to ensure that all valves had ample time to open and close fully and to maximize the volume pumped per

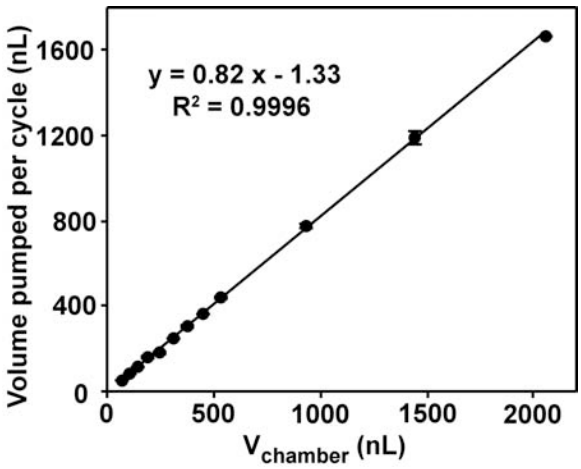


Figure 2.5: Maximum volume of water pumped per cycle as a function of the diaphragm valve chamber volume for pumps 1 through 12. Valve actuation vacuum and pressure were −80 and 40 kPa, respectively. A three-layer device with a 20 μm etch depth fluid layer, 70 μm etch depth manifold layer, and 254 μm thick HT-6135 PDMS membrane was used. Diaphragm valve displacement chamber dimensions are presented in Table 1; the input and output valves of each pump were held constant at $V_{chamber} = 274$ nl. For all valves, $w_{expansion} = 300$ μm, $l_{expansion} = 500$ μm, and the gap was 500 μm.

cycle. The linear correlation shows that the maximum volume pumped per cycle is directly dependent upon $V_{chamber}$, with approximately 82% of $V_{chamber}$ pumped per cycle. This relationship between volume pumped per cycle and displacement chamber volume enables the design of pumps for metering precisely known volumes.

Figure 2.6 explores the relationship between diaphragm valve actuation time and

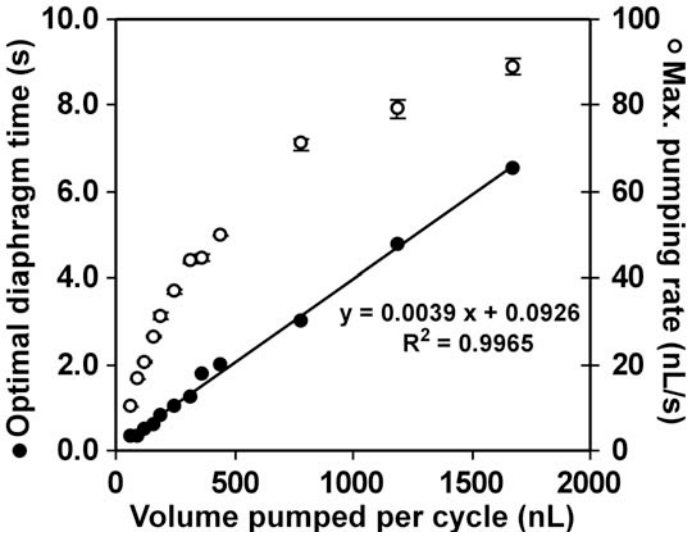


Figure 2.6: Optimal diaphragm actuation time (•) and maximum pumping rate (∘) as functions of volumes of water pumped per cycle for pumps 1 through 12. Device parameters were the same as in Figure 2.5.

volume pumped per cycle for pumps 1 through 12 at zero pressure head. The optimal diaphragm actuation time of each pump was determined by holding cycle steps 1, 3, and 4 at an excess actuation time of 1.5 s each and varying the actuation time of the diaphragm valve (steps 2 and 5) until a maximum pumping rate was reached. The optimal diaphragm valve actuation time was found to be a linear function of the volume pumped per cycle, indicating that regardless of pump size, diaphragm

valve emptying (the actual "pumping step" in the cycle) occurred at a constant rate of 260 nl/s determined from the reciprocal of the slope of the linear regression. A constant fluid flow rate is expected in a system with constant pressure (the 40 kPa valve actuation pressure) forcing fluid through a channel with constant cross-sectional area ($A_{fluidic} = 1030\ \mu\text{m}^2$); our data clearly fit this expectation.

Figure 2.6 also explores the maximum pumping rate attainable for each pump at zero pressure head. At the smallest volume pumped per cycle (pump 12), only 300 ms of the total 5.1 s cycle (6%) was spent emptying the diaphragm valve and the overall pumping rate was only 10 nl/s. At the largest volume pumped per cycle (pump 1), 6.5 s of the total 17.5 s cycle (37%) was used for closing the diaphragm valve and the pumping rate rose to 89 nl/s. To pump at this rate, the diaphragm valve emptied at an overall rate of 240 nl/s. This value is close to the 260 nl/s maximum measured earlier and indicates that the pump operation is indeed optimized. Other results indicate that reducing the actuation times for steps 1, 3, and 4 (which were kept excessively long for the purposes of this figure) increases both the fraction of the cycle devoted to diaphragm emptying and the overall pumping rate. Also, since smaller valves require shorter actuation times and $V_{chamber}$ of the input and output valves does not have a direct effect on the volume pumped per cycle, minimizing $V_{chamber}$ of the input and output valves further increases the overall pumping rate.

Figure 2.7 explores the effect of fluidic channel cross-sectional area $A_{fluidic}$ on the optimal cycle time using pumps 25, 28, 30, 33, 35, and 38. The optimal cycle

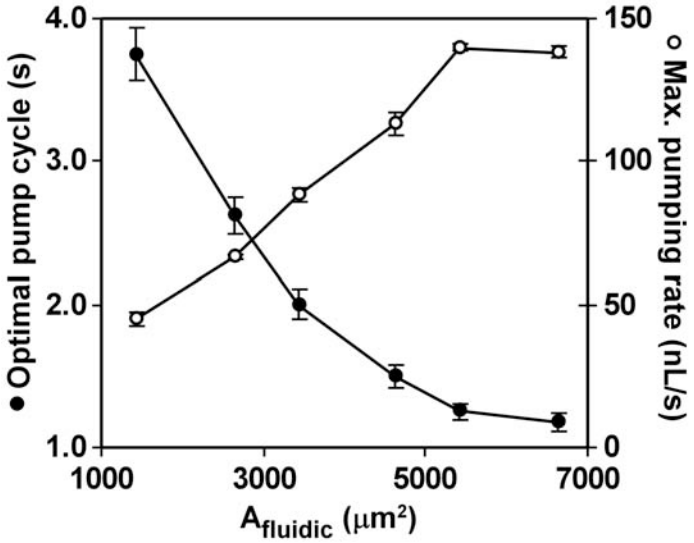


Figure 2.7: Optimal pump cycle time (•) and maximum water pumping rate (○) as functions of $A_{fluidic}$ for pumps 25, 28, 30, 33, 35, and 38. Valve actuation vacuum and pressure were −80 and 40 kPa, respectively. Channel cross-sectional areas are presented in Table 1; other device parameters were the same as in Figure 2.5

time of each pump was found by setting all five cycle steps to the same time and varying this time until a maximum pumping rate was reached. The inverse second-order polynomial relationship between optimal pump cycle time and $A_{fluidic}$ is in agreement with Poiseuille's law using a constant pressure to force a constant volume of fluid through cylindrical channels of different cross-sectional areas. Figure 2.7 also shows the maximum pumping rate for pumps 25, 28, 30, 33, 35, and 38. The maximum pumping rate attainable rises as a roughly linear function of the fluid channel cross-sectional area until a maximum pumping rate of approximately 140 nl/s is reached. This plateau is likely due to the fast cycle rate coupled with the finite amount of time required to pressurize and depressurize the pneumatic control

system. This relationship between fluidic channel cross-sectional area and maximum pumping rate makes pumps designed for specific pumping rates possible.

Finally, Figure 2.8 explores the effect of cycle time for pumping water against

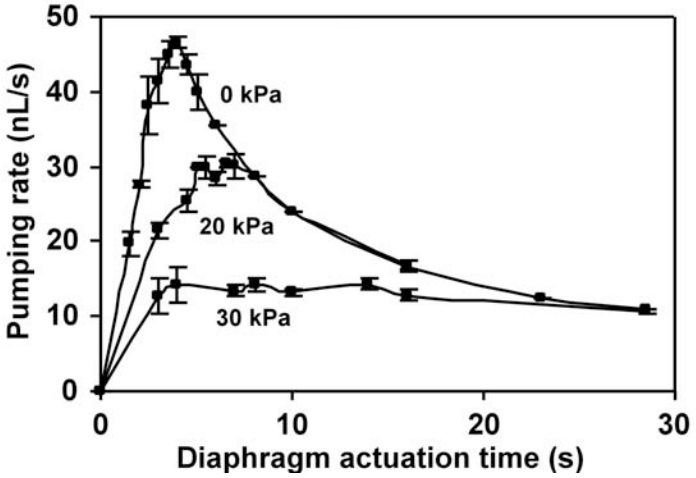


Figure 2.8: Pumping rates attainable at three different pressure heads as functions of diaphragm valve actuation time. Pump 3 on a three-layer device with a 20 μm etch depth fluid layer, 70 μm etch depth manifold layer, and 254 μm thick HT-6240 PDMS membrane was used. Valve actuation vacuum and pressure were -80 and 40 kPa, respectively.

various regulated pressure heads at the output. Pumping rate data were obtained at pressure heads of 0, 20, and 30 kPa using pump 3 in Figure 2.2. Only diaphragm valve actuation times (steps 2 and 5) were varied; input and output valve actuation times (steps 1, 3, and 4) were held at a constant dwell time of 1.5 s each. At 0 kPa pressure head, a clear maximum pumping rate of 47 nl/s was reached at a diaphragm actuation time of 5 s. This maximum pumping rate is lower than the 70 nl/s measured for this pump in Figure 6 because of the decreased elasticity of the HT-6240 membrane (250% elongation) compared to the HT-6135 membrane (450% elongation) used in

prior pump characterizations. These and other results indicate that identical pumps fabricated with thicker or less-elastic PDMS membranes consistently pump a smaller fraction of $V_{chamber}$ per cycle because the membrane fills less of the displacement chamber and translates less of $V_{chamber}$ to the fluidic wafer, and pumps fabricated with thinner or more-elastic membranes pump a larger fraction of $V_{chamber}$ per cycle as the membrane fills more of the displacement chamber. Applying a pressure head decreased the maximum pumping rate, with 30 nl/s attainable at 20 kPa pressure head and 13 nl/s attainable at 30 kPa pressure head. Pumping rates at all three pressure heads converge when diaphragm valve actuation times are used that exceed the time required for complete closure of the diaphragm valve. While pumping rates decreased with applied fluid pressure, reliable pumping was attainable at fairly high pressure heads. In a related study, the maximum reachable pressure at blocked flow was measured by blocking the output of a diaphragm pump and actuating the pump until the output pressure reached a constant. As expected, the output pressure was found to asymptotically approach the pump actuation pressure, with 42 kPa output pressure attainable using a 43 kPa actuation pressure.

2.4 Discussion and conclusions

The monolithic valves and pumps developed and evaluated here have a number of advantages for nl- and μl-scale fluid manipulation. Monolithic membrane valves are "normally closed" and require no manifold pressure when sealing against the neg-

ligible fluid pressures commonly encountered in many microfluidic devices. Devices utilizing "normally open" pneumatic valves [84] cannot be depressurized without losing control of the fluid contents. The monolithic membrane valves presented here are larger than some pneumatic valves in the literature [83, 84] and provide a relatively large active area for actuation pressure and vacuum. This decreases the magnitude of pressure and vacuum required to actuate the valves and increases the maximum fluid pressure against which the valves seal without failure: 10 kPa manifold pressure seals the monolithic membrane valves against fluid pressures up to 40 kPa, and ∼20 kPa manifold vacuum is adequate to fully open the valve. While smaller pneumatic valves can be fabricated in denser arrays and actuated more rapidly, they require greater pressures [84] or vacuums [83] for reliable actuation. The four-layer monolithic membrane valves contribute 20 nl dead volume and the smallest three-layer valves contribute only 8 nl dead volume; these volumes are an order of magnitude smaller than the analyte volumes commonly encountered in many current microfluidic bioassays [68, 96]. The use of commercially available elastomer membranes in the valves and pumps simplifies and expedites device fabrication. The monolithic membrane diaphragm pumps have been demonstrated to pump reliably at a variety of pumping rates and pressure heads. They are self-priming and pump fluids forward or backward simply by reversing the actuation cycle. Indeed, any number of input/output valves may be connected to a single diaphragm valve to construct a multidirectional fluidic router. The integrated microfabricated manifold like that used in

all-elastomer devices [83, 84] allows for monolithic membrane valve and pump placement at any point on the analysis device and actuation of arrays of pumps or valves in parallel. Finally, by adjusting the volume of the diaphragm valve displacement chamber, the volume pumped per actuation may be determined at the fabrication stage. Diaphragm pumps may therefore be used to meter nanoliters to microliters of fluid in applications that require precise control of fluid volumes and fluid position within a device.

In conclusion, reliable microvalves and micropumps suitable for large-scale integration into glass chemical and biochemical assay devices have been fabricated and characterized. Facile microfabrication coupled with an integrated manifold make massively parallel actuation of arrays of valves and pumps possible for the first time in glass microfluidic devices. In addition, the four-layer valve and pump design incorporates an all-glass fluidic system to minimize fluid-PDMS contact for improved chemical compatibility. Systematic analysis of dimensions and actuation conditions shows that valves and pumps with specific operational characteristics can be easily designed and fabricated. The simplicity of large-scale monolithic valve and pump fabrication coupled with the chemical compatibility of the glass microfluidic analysis platform make these valves and pumps well suited for integration into bioassay devices. For example, this work will be critical in the development of arrays of PCR-CE integrated microdevices following the methods of Lagally *et al.* [68,95,96] and the development of complete microfabricated chemical analysis systems for extraterrestrial

exploration [98].

Acknowledgments

Device fabrication was performed at the University of California, Berkeley Microfabrication Laboratory. This research was supported by the Director, Office of Science, Office of Biological and Environmental Research of the US Department of Energy under Contract DEFG91ER61125. AMS is supported by NASA grant NAG5-9659. ETL gratefully acknowledges the support of a Whitaker Foundation Predoctoral Fellowship.

Chapter 3

An Integrated Microfluidic Processor for Single Nucleotide Polymorphism-based DNA Computing

Reproduced by permission of The Royal Society of Chemistry from "An Integrated Microfluidic Processor for Single Nucleotide Polymorphism-based DNA Computing" by William H. Grover and Richard A. Mathies, *Lab on a Chip* **5**, 2005 (1033-1040).

An integrated microfluidic processor is developed that performs molecular computations using single nucleotide polymorphisms (SNPs) as binary bits. A complete population of fluorescein-labeled DNA answers is synthesized containing three distinct polymorphic bases; the identity of each base (`A` or `T`) is used to encode the value of a binary bit (TRUE or FALSE). Computation and readout occur by hybridization to complementary capture DNA oligonucleotides bound to magnetic beads in the microfluidic device. Beads are loaded into sixteen capture chambers in the processor and suspended in place by an external magnetic field. Integrated microfluidic valves and pumps circulate the input DNA population through the bead suspensions. In this example, a program consisting of a series of capture/rinse/release steps is executed and the DNA molecules remaining at the end of the computation provide the solution to a three-variable, four-clause Boolean satisfiability problem. The improved capture kinetics, transfer efficiency, and single-base specificity enabled by microfluidics make our processor well-suited for performing larger-scale DNA computations.

3.1 Introduction

Ten years after the invention of the transistor, microfabrication methods were devised to efficiently pack transistors onto silicon chips, and the now-ubiquitous integrated circuit was born. In 1994 Adleman used DNA to perform the first molecular computation [11]. This and subsequent implementations of molecular computing would similarly benefit from the development of a molecular integrated circuit—a device to automate, miniaturize, and parallelize molecular computing operations. An integrated circuit for DNA computation was first described conceptually in 1999 as an array of microreactors with DNA oligonucleotides immobilized in the reactors for sequence-specific capture and redirection of input DNA, together with a microfluidic pumping mechanism for routing DNA solutions between the reactors [45]. In this hypothetical computer, input oligonucleotide populations encoding all possible answers to a given problem would be introduced into the device, a series of sequence-specific capture and release operations would separate oligonucleotides encoding incorrect answers from those encoding correct ones, and any input oligonucleotides remaining at the end of the computation would encode the correct solution or solutions to the problem. Execution of this vision was difficult in 1999 because the available microfluidic valve, pump, and capture technologies were chemically or physically unsuitable for use with DNA or were difficult to fabricate in the dense arrays required for the molecular computer. [60, 71, 76, 77, 79, 99]

Significant progress has subsequently been made using beads trapped in microflu-

idic devices for sequence-specific capture of oligonucleotides in computational [50, 56] and in analytical [100–105] applications. However, since typical structures lack active fluid control, only a single pass of an input DNA solution through the beads is possible, and the ability of these devices to successfully capture and route DNA is limited. In part to compensate for poor capture kinetics, many demonstrations of hybridization-based DNA computing have used long hybridization times and multiple-base capture sequences to represent single binary bits. The most complex DNA computation performed to date was solved using 300 base pair input DNA containing constant 15-base sequences for each of 20 binary bits [29]. The computation required 4 hours for each of 24 hybridization steps for a total runtime of 96 hours. Recent advances in DNA microarray technology clearly demonstrate that constant multi-base capture sequences and extensive hybridization times are not in principle required for reliable molecular recognition [106].

Recently, we developed a novel pneumatic membrane valve and pump design that enables the fabrication of dense, large-scale arrays of actuators on glass microfluidic devices [107]. These valve technologies have thus far been instrumental in the development of integrated genetic analyzers for DNA sequencing and analysis [108], pathogen and infectious disease detectors [109], and amino acid analysis systems for space exploration [110]. Here we exploit this pneumatic membrane valve and pump technology to develop and demonstrate a microfluidic processor for performing DNA computations using single nucleotide polymorphisms (SNPs) as binary bits.

Our processor interrogates an 11-base long fluorescein-labeled input DNA population that includes three polymorphic bases for a total of eight unique sequences. These eight sequences represent all possible solutions to a three-bit computation, with the actual base at each polymorphism (`A` or `T`) representing a binary value (TRUE or FALSE) for the bit associated with the polymorphism. The processor shown in Figure 3.1 includes an array of 16 capture chambers (A–P) containing magnetic beads derivatized with biotinylated capture oligonucleotides complementary to specific members of the input population. An integrated monolithic membrane diaphragm pump and 32 bus valves are used to circulate input oligonucleotides through the bead suspensions in selected capture chambers. Correct answers (perfectly-complementary oligonucleotides) are captured by hybridization, and incorrect answers are eliminated by rinsing the bead suspension. Oligonucleotides remaining after a series of capture/rinse/release steps represent correct solutions to the logical satisfiability problem encoded in the path followed by the oligonucleotides through the device. The enhanced capture and transfer efficiency provided by microfluidics makes possible this first demonstration of a hybridization-based DNA computation using SNPs to represent binary bits.

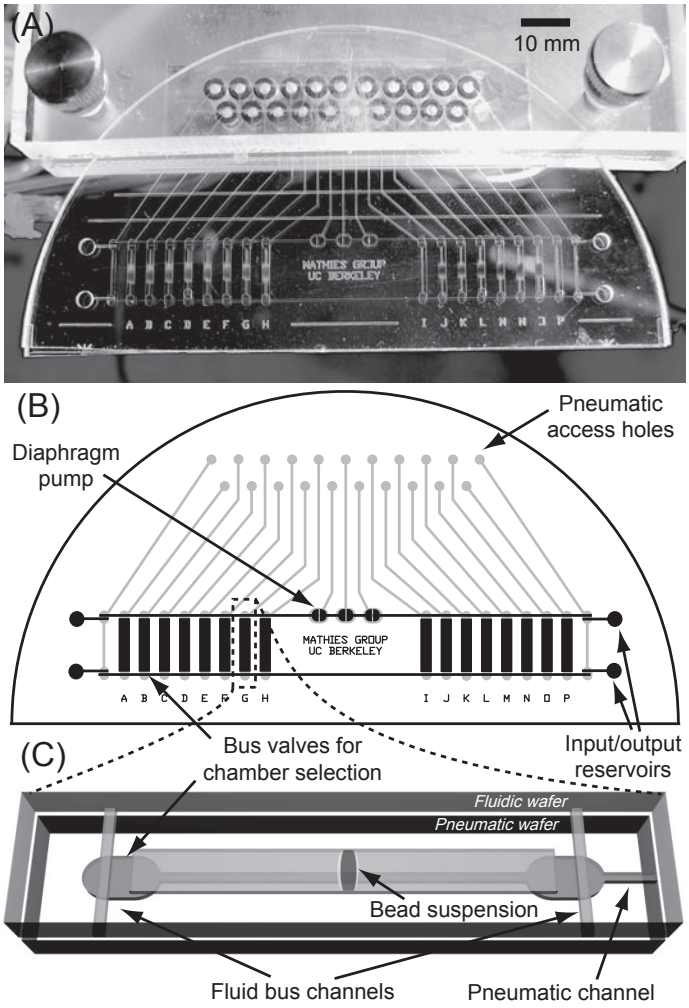


Figure 3.1: Photograph (A) and mask design (B) of the microfluidic processor. Pneumatic (grey) and fluidic (black) features were etched into a single 10 cm diameter, 1.1 mm thick glass wafer that was subsequently split into the fluidic and pneumatic layers and bonded reversibly with the PDMS membrane. Twenty-one pneumatic access holes supply actuation vacuum (−60 kPa) and pressure (10 kPa) to the three pump valves, 32 bus valves, and four input/output valves. (C) Oblique view of a single capture chamber. When vacuum is applied to a pneumatic channel, the bus valves open as the PDMS membrane pulls away from the fluidic wafer and flow in the fluid bus is diverted through the capture chamber. Magnetic beads (12 μg per chamber) are held in place by rare earth magnets placed above and below the processor (not shown). Each chamber contains 1.1 μL of fluid; a typical active volume of the device while circulating (chambers A and O plus the volume contained in the fluidic loop and pump valves) is 5.8 μL.

3.2 Methods

3.2.1 Device fabrication

Device features (see Figure 3.1) were etched into glass wafers using conventional photolithography and wet chemical etching [111]. Briefly, 1.1 mm thick, 100 mm diameter borosilicate glass wafers were coated with 200 nm of polysilicon using low-pressure chemical vapor deposition. The wafers were then spin-coated with positive photoresist, soft-baked, and patterned with the device design using a contact aligner and a chrome mask. After development and removal of irradiated photoresist, the exposed polysilicon regions were removed by etching in SF_6 plasma and the exposed regions of glass were etched isotropically in 49% HF to a depth of 100 μm. After stripping the remaining photoresist and polysilicon layers, the wafers were drilled with 1.1 mm diameter holes for pneumatic connections and 2.5 mm diameter holes for fluidic reservoirs. The wafers were then scored and broken, and the resulting fluidic and pneumatic layers were bonded using a 254 μm thick polydimethylsiloxane (PDMS) elastomer membrane [107]. Internal surfaces of the device were treated with a 5% bovine serum albumin solution before use to reduce adsorption of DNA.

3.2.2 Basis for computation

Computing with single DNA bases involves two related kinds of complementarity between the immobilized capture and fluorescent input DNA populations: base-wise

(T and A) and bit-wise (TRUE and FALSE) complementarity. Arbitrarily, T represents TRUE and A represents FALSE for a given bit B_n in the biotinylated (Bio) capture population 5'-Bio-agB_0tcB_1caB_2gt-3', and A represents TRUE and T represents FALSE for the same bit in the fluorescein-labeled (FAM) input population 5'-FAM-acB_2tgB_1gaB_0ct-3'. Capture by hybridization occurs only if the input and capture oligonucleotides are perfectly complementary, meaning that the bit values encoded in the input oligonucleotide satisfy all the logical requirements of the capture oligonucleotide. After a series of capture/rinse/release steps, input oligonucleotides remaining at the end of the series encode a solution that satisfies the conjunction (the Boolean AND) of the requirements at each individual step. If two or more different capture oligonucleotides are bound to beads in the same chamber, input oligonucleotides captured in the chamber encode a solution that satisfies the disjunction (the Boolean OR) of the requirements of the different capture oligonucleotides. Finally, capture oligonucleotides containing A at bit B_n capture only negated values of that bit (the Boolean NOT) [112]. Using this method and our device, we set out to solve the 3-bit, 4-clause Boolean satisfiability problem [NOT(B_0) OR B_2] AND [B_0 OR B_1] AND [NOT(B_1) OR NOT(B_2)] AND [NOT(B_0) OR B_1], which is TRUE only if B_0 = FALSE, B_1 = TRUE, and B_2 = FALSE.

3.2.3 Capture and input DNA populations

Six synthesis runs of 11-base biotinylated oligonucleotides provided the populations of capture oligonucleotides used in the computation (5'-Bio-agTtcWcaWgt-3', Bio-agWtcTcaWgt, Bio-agWtcWcaTgt, Bio-agAtcWcaWgt, Bio-agWtcAcaWgt, and Bio-agWtcWcaAgt; W indicates either A or T and polymorphic bases are capitalized). To prepare each of the bead-oligonucleotide conjugates, 5 nmol of biotinylated oligonucleotides (Integrated DNA Technologies, Coralville, IA) and 85 μg of 1.5 μm diameter streptavidin-coated magnetic beads (Bangs Laboratories, Fishers, IN) were incubated in 50 μL TTL buffer (100 mM Tris-HCl, 0.1% Tween 20, 1.0 M LiCl) at room temperature for 3 h. The derivatized beads were then rinsed in 0.15 N NaOH to eliminate nonspecifically-bound oligonucleotides, then rinsed twice in TT buffer (250 mM Tris-HCl, 0.1% Tween 20), and finally incubated in TTE buffer (250 mM Tris-HCl, 0.1% Tween 20, 20 mM EDTA) at 80 °C for 10 min to remove any remaining unstable linkages. A single 11-base synthesis run of the fluorescein-labeled 5-FAM-acWtgWgaWct-3 provided the input DNA population for the computation and two additional syntheses (FAM-acWtgAgaWct and FAM-acWtgTgaWct) yielded the populations used in the characterization studies. Oligonucleotide sequences and experimental conditions were chosen to maximize the difference between the calculated melting temperatures of the least-stable perfectly-matched duplex (T_m = 47.3 °C) and the most-stable single-base-mismatched duplex (T_m = 37.8 °C), ensuring that capture occurs only between perfectly-complementary oligonucleotides [113]. The sequences and compu-

tational roles of the oligonucleotides are summarized in Table 3.1.

Synthesis ($5' \rightarrow 3'$)	Population members ($5' \rightarrow 3'$)	Computational role
Bio-agTtcWcaWgt (B_0 = TRUE)	Bio-agTtcTcaTgt Bio-agTtcTcaAgt Bio-agTtcAcaTgt Bio-agTtcAcaAgt	captures B_0 = TRUE (FAM-acWtgWgaAct)
Bio-agWtcTcaWgt (B_1 = TRUE)	Bio-agTtcTcaTgt Bio-agTtcTcaAgt Bio-agAtcTcaTgt Bio-agAtcTcaAgt	captures B_1 = TRUE (FAM-acWtgAgaWct)
Bio-agWtcWcaTgt (B_2 = TRUE)	Bio-agTtcTcaTgt Bio-agTtcAcaTgt Bio-agAtcTcaTgt Bio-agAtcAcaTgt	captures B_2 = TRUE (FAM-acAtgWgaWct)
Bio-agAtcWcaWgt (B_0 = FALSE)	Bio-agAtcTcaTgt Bio-agAtcTcaAgt Bio-agAtcAcaTgt Bio-agAtcAcaAgt	captures B_0 = FALSE (FAM-acWtgWgaTct)
Bio-agWtcAcaWgt (B_1 = FALSE)	Bio-agTtcAcaTgt Bio-agTtcAcaAgt Bio-agAtcAcaTgt Bio-agAtcAcaAgt	captures B_1 = FALSE (FAM-acWtgTgaWct)
Bio-agWtcWcaAgt (B_2 = FALSE)	Bio-agTtcTcaAgt Bio-agTtcAcaAgt Bio-agAtcTcaAgt Bio-agAtcAcaAgt	captures B_2 = FALSE (FAM-acTtgWgaWct)
FAM-tcWagWgtWca (all 8 possible values for B_2, B_1, and B_0)	FAM-acTtgTgaTct FAM-acTtgTgaAct FAM-acTtgAgaTct FAM-acTtgAgaAct FAM-acAtgTgaTct FAM-acAtgTgaAct FAM-acAtgAgaTct FAM-acAtgAgaAct	input population (FAM-acWtgWgaAct)

Table 3.1: Sequences and computational roles of oligonucleotides. Polymorphic bases used for bit encoding are capitalized. W in a synthesis run indicates either T or A. Bio and FAM indicate 5' biotin and fluorescein oligonucleotide modifications, respectively.

3.2.4 Programming the integrated circuit

The integrated circuit is programmed by loading the desired combinations of beads labeled with capture oligonucleotides into the various chambers on the device. Two sets of bus valves are opened to select the target chamber on the right side of the device and the input/output reservoirs on the left side. A suspension of beads (5 μL of a 2.5 μg/μL solution) in hybridization buffer (700 mM NaCl, 10% formamide, 10 mM phosphate pH 7.2, 0.1% Tween-20) is pumped from the left input reservoir through the selected chamber on the right side of the device, where they are captured by

magnets placed above and below the chamber (Figure 3.2A). The process is reversed

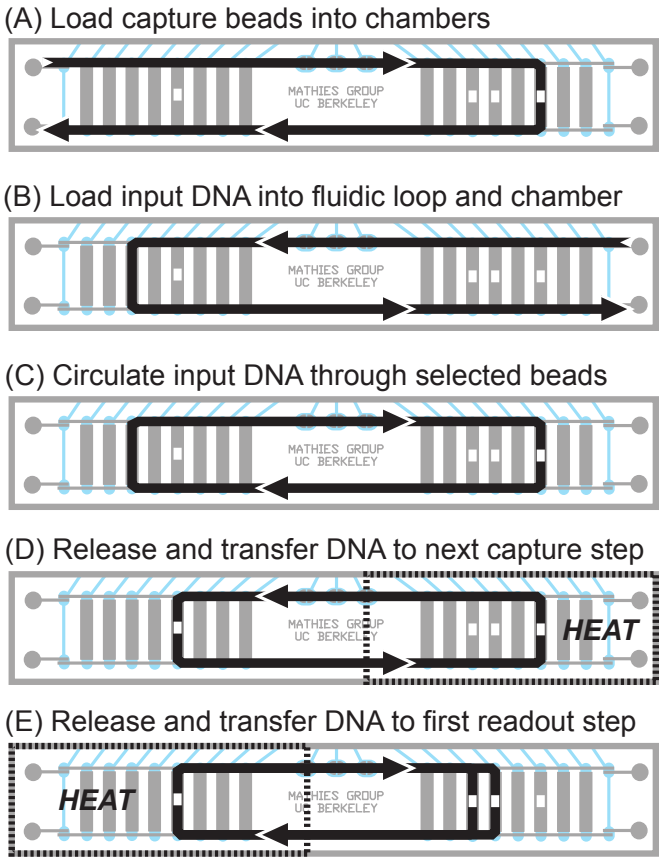


Figure 3.2: Diagrams showing fluid flow during loading (A and B), capture (C), release and recapture (D), and readout (E) steps of the microfluidic processor.

(right input reservoirs to left chamber) when filling chambers on the left side of the device. The trapped beads are then rinsed by pumping 10 μL of fresh hybridization buffer through the bead suspension, and the loading process is repeated for each bead chamber. Next, a single load chamber is filled with a 30 μM solution of the fluorescent input population in hybridization buffer (Figure 3.2B). Input solution is loaded until the entire volume of the load chamber plus the fluidic loop and three

open pump valves is filled with the solution. This volume (5.8 μL) sets the amount of input DNA going into the first step of the computation (170 pmol), which is nine times greater than the experimentally-determined capacity of the beads in the first capture chamber.

3.2.5 Performing the computation

In the first step of a computation, two sets of bus valves are opened to join the load chamber on the left side of the device and first capture chamber on the right side in the microfluidic circuit. With the magnet holding the bead suspension in place and the beads maintained at 25 °C, the 5.8 μL fluid contents of the device are pumped in a clockwise circuit through the load and capture chambers (Figure 3.2C). The three pump valves are actuated according to a six-step, three second pumping pattern [107]. After 30 minutes of pumping (roughly 22 passes of the input oligonucleotide solution through the bead suspension), the load chamber is rinsed with buffer from the right input reservoir and the beads in the capture chamber are rinsed from the left input reservoir to eliminate incorrect (mismatched) input oligonucleotides. The beads are then collected against the edge of the chamber using the magnet and the uniform bed of beads is imaged using an inverted fluorescence microscope (Nikon) with excitation light from a mercury arc lamp passed through a 480 nm band-pass filter and reflected off a 505 nm long-pass dichroic beamsplitter through a 4x objective lens; the measured power at the objective was 17 mW. Fluorescence was collected

back through the objective and passed through the dichroic and a 535 nm band-pass filter before impinging upon a CCD detector (QImaging, Burnaby, BC, Canada) for a 2 s exposure.

In the second step, the right-side bead chamber that captured input oligonucleotides in the first step is heated using an external heater to 50 °C (2 °C higher than the mean calculated T_m of the eight perfect complements) while maintaining the new second bead chamber on the left side of the device at 25 °C. Two sets of bus valves are opened to join the hot and cool chambers in the circuit, and the released input oligonucleotides are pumped counterclockwise between the old and new chambers (Figure 3.2D). After 30 minutes of pumping, the chambers are again rinsed and imaged as described above. This back-and-forth process is then repeated for all computation chambers, with additional incorrect answers eliminated at each step.

To readout the result of a computation, input oligonucleotides captured in the last step of the computation are simultaneously transferred to two chambers, one containing beads capturing `A` (TRUE) for a given base/bit and the other capturing `T` (FALSE) (Figure 3.2E). A special 45 min pumping cycle alternates flow between the two destination chambers every three seconds, and the beads are not rinsed after readout captures because no elimination of incorrect answers is occurring. The value of the bit is determined by comparing the relative fluorescence of the two chambers, and the overall solution to the problem is read by repeating the readout process for the other two bits.

3.3 Results

3.3.1 Bit fidelity and rinsing efficiency

To demonstrate the feasibility of using single-base polymorphisms to encode binary bits, chambers B and O were loaded with 12 μg of 5'-Bio-`agWtcTcaWgt`-3' and 5'-Bio-`agWtcAcaWgt`-3' beads, respectively. Two separate single-step captures were performed by filling the loop volume each time with an 10× excess of 5'-FAM-`acWtgAgaWct`-3' (5.8 μL of a 30 μM solution or 170 pmol) and using the on-chip diaphragm pump to circulate the solution through the beads for 30 min. Figure 3.3 compares background-corrected fluorescence images of beads in the two chambers at various times in the post-capture bead rinse. Every immobilized capture oligonucleotide in chamber B had a perfect complement in the fluorescent input population, and the resulting bead fluorescence started high and decreased slowly with rinsing. In contrast, every capture oligonucleotide in chamber O had at least a single base mismatch with every input oligonucleotide, and the observed fluorescence intensity started low and decreased quickly as mismatched input oligonucleotides were rinsed away. The ratio of mismatched to perfectly-complementary fluorescence (O/B) is a measurement of computational error introduced by incomplete capture of perfectly-matched input oligonucleotides and partial capture of mismatched input oligonucleotides. Rinsing the beads for three minutes reduced the error rate to 7.6%, and five additional minutes of rinsing further reduced the error rate to only 1.1%.

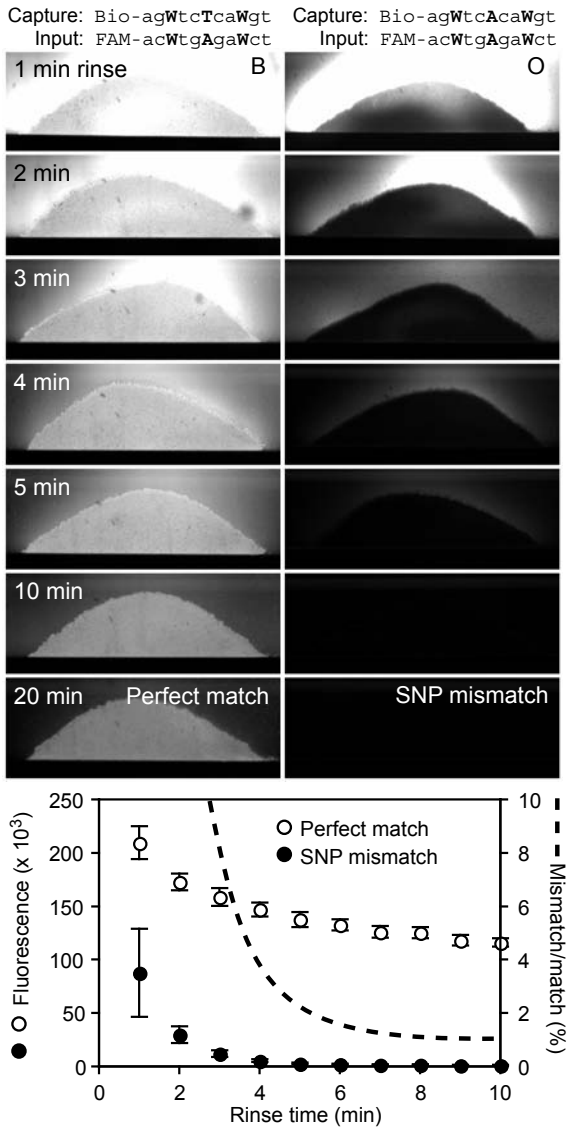


Figure 3.3: Fluorescence images of capture beads in the bit fidelity and rinsing efficiency experiment. The sustained bead fluorescence with rinsing in chamber B confirms capture of the perfectly-complementary input population, while the rapid elimination of fluorescence with rinsing in chamber O verifies that single-base mismatches prevent capture. In the plot of fluorescence counts at various rinse times, the ratio of the single-base mismatch fluorescence (closed circles) to the perfectly-complementary fluorescence (open circles) provides an estimate of computational error associated with a single capture step (dashed line); an 8 min rinse reduces the error to 1.1%. Fluorescence count data was obtained by integrating over a region of constant area in each background-corrected image. Error bars indicate the standard deviation among individual pixel fluorescence values due to the spatial inhomogeneity within each bead bed.

Other results indicated that protocols utilizing more stringent hybridization buffers (less salt or more formamide) increased mismatch detection sensitivity at shorter rinse times, at the expense of decreased step-to-step transfer efficiency.

In a separate calibration experiment, after an eight minute rinse, the oligonucleotides captured in chamber B were heated and released into an on-chip loop of known volume. The fluorescence intensity of this solution was compared with intensities from a series of standards to determine that 18 pmol of the initial 170 pmol of input oligonucleotides were captured in chamber B. This result did not change significantly with the use of more concentrated input oligonucleotide solutions, indicating that in this and subsequent experiments the 18 pmol capacity of the first capture/rinse/release step determined the total amount of input DNA used in the device.

3.3.2 Serial capture/rinse/release efficiency

Figure 3.4 presents the program used to determine the efficiency of serial capture and release on the microfluidic processor. Chambers A and O each contained 12 μg of Bio-`agWtcTcaWgt` beads; chamber P and the loop volume were loaded with an 10x excess of FAM-`acWtgAgaWct` (5.8 μL of a 30 μM solution or 170 pmol). Since each oligonucleotide in the input population had an exact complement in the capture population, no elimination of mismatched input DNA occurred in this experiment. The input oligonucleotides were transferred back and forth eight times between chambers

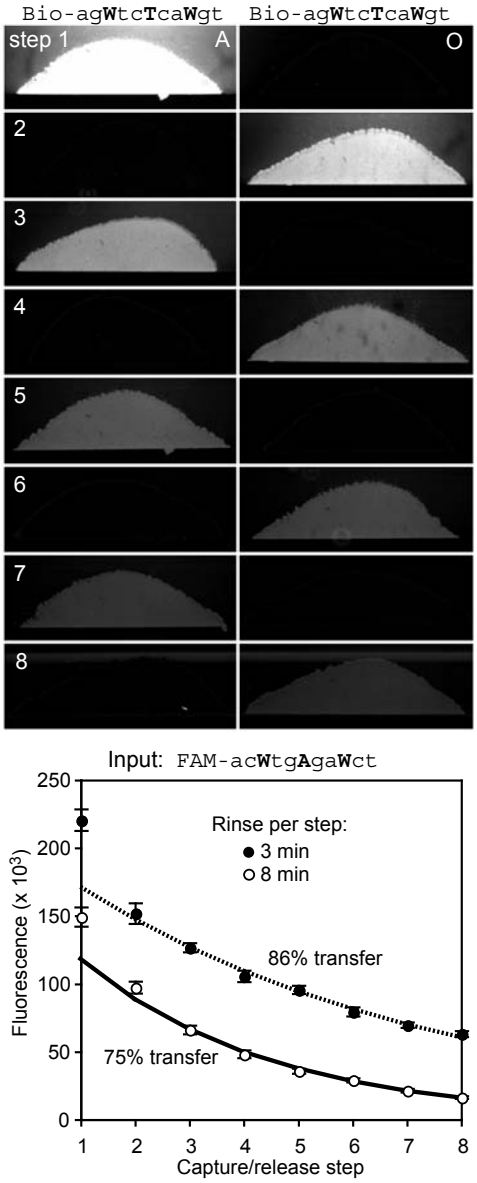


Figure 3.4: Fluorescence images of the capture beads in both chambers at each step in the capture/release efficiency measurement, with 30 min capture and 8 min rinse after each capture/release step and a 3 s exposure. The back and forth pattern of fluorescence confirms the sequential transfer of perfectly-complementary input oligonucleotides between chambers A and O. In the accompanying plot of fluorescence counts from the capture chambers, using the more-stringent 8 min rinse after each capture step (open circles) results in a 75% step-to-step transfer efficiency (solid line), while a less-stringent 3 min rinse (closed circles) increases the transfer efficiency to 86% (dotted line).

A and O, with 30 minutes per capture/rinse/release step and either three or eight minute rinse times after each capture. A measured pumping rate of 70 nL/s and a constant 5.8 μL loop volume indicate that during each 30 min capture/rinse/release step the contents of the loop were passed through the beads 22 times. Figure 3.4 shows fluorescent images of beads in chambers A and O after each of the eight steps. The more-stringent eight minute rinse resulted in 75% transfer of input oligonucleotides from one step to the next, while the less-stringent three minute rinse transferred 86% of input oligonucleotides from step to step. Additional experiments confirmed that photobleaching caused by repetitive imaging of the same input population had a negligible effect on the measured fluorescence intensity; protocols using less stringent hybridization conditions increased transfer efficiencies to nearly 99% but were unsuitable for single-base mismatch detection.

For both rinse times, the amount of input DNA lost when transferring between steps 1 and 2 was larger than predicted by the calculated transfer efficiencies; a smaller loss was observed using the longer eight minute rinse time. This loss was attributed to excess nonspecific binding of input DNA to the beads that was more significant at the high input oligonucleotide concentrations present in the first capture step (30 μM) and less significant at the lower input oligonucleotide concentrations in the second (3.2 μM) and subsequent steps. For this reason, subsequent experiments were performed using an eight minute rinse (1.1% error) in the first capture/rinse/release step and a three minute rinse (7.6% error) in subsequent steps.

3.3.3 Satisfiability computation

In the satisfiability computation, 170 pmol of the input population FAM-acW-tgWgaWct was loaded into the 5.8 μL loop volume and subjected to four serial capture/rinse/release steps representing the four clauses in the satisfiability problem. Figure 3.5 presents fluorescence images of beads in the four computational cham-

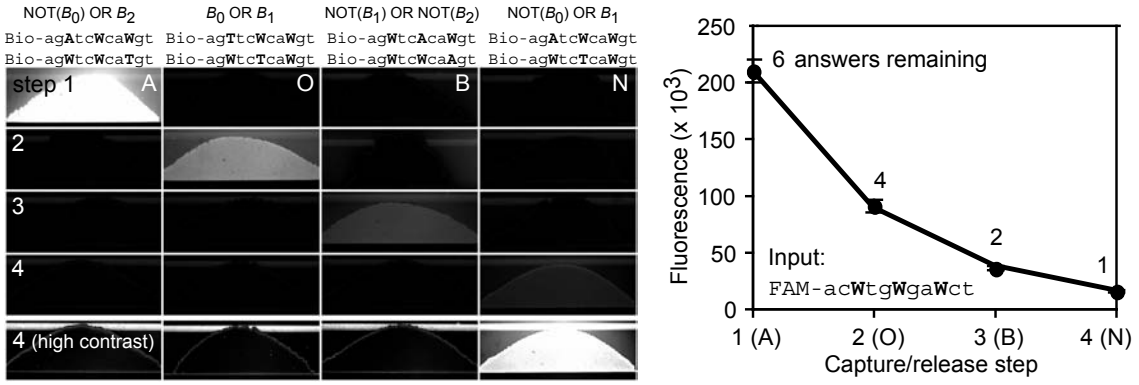


Figure 3.5: Fluorescence images (3 s exposures) of beads in the four computation steps involved in the solution of the problem [NOT(B_0) OR B_2] AND [B_0 OR B_1] AND [NOT(B_1) OR NOT(B_2)] AND [NOT(B_0) OR B_1]. The decrease in fluorescence from step 1 to step 4 corresponds to the elimination of the seven different incorrect answers to the problem. The enhanced contrast images of step 4 show that the input oligonucleotides representing the correct answer to the problem have been captured uniquely in chamber N. In the plot of fluorescence intensity in the computing steps, the solid line traces a calculated elimination of half of the fluorescence intensity per step (as expected from the computation) plus an additional 8% loss (92% step-to-step transfer efficiency), which matches the observed data points accurately.

bers A, O, B, and N, at each of the four computation steps. The immobilized DNA contents and the computational role of each chamber are also indicated above each column. The loss of fluorescence observed as the initial 18 pmol of input oligonucleotides were transferred from A to O to B to N (30 min and 22 passes through the

bead suspensions per transfer) corresponds to the elimination of 16 pmol of seven oligonucleotides encoding incorrect answers to the problem. Fluorescence counts decreased at a rate of 50% per step combined with a 92% transfer efficiency. In step 1, the input oligonucleotides FAM-`acAtgTgaTct` and FAM-`acAtgAgaTct` were captured by members of both Bio-`agAtcWcaWgt` and Bio-`agWtcWcaTgt` in chamber A. This increased the measured fluorescence in step 1 but had no effect on the outcome of the computation because FAM-`acAtgTgaTct` and FAM-`acAtgAgaTct` were eliminated completely in steps 2 and 3, respectively.

Three additional capture/release steps were then used to read out the identity (`A` or `T`) of each polymorphic base and the corresponding value (TRUE or FALSE) of each bit in the oligonucleotides captured in chamber N. The oligonucleotides were released and transferred to the first readout step by heating chamber N while pumping the oligonucleotides from N through chambers C and D simultaneously. Immobilized capture oligonucleotides in chambers C and D differed only at a single polymorphic base (`A` or `T`) at bit B_0, and the identity of the complementary base in the remaining input oligonucleotides (`T` or `A`) determined which one of the two chambers captured the oligonucleotides. This process was then repeated twice for the two remaining bits, using chambers L and M to read out the value of B_1 and E and F to read out B_2. Figure 3.6 presents fluorescence images of the beads in the three readout steps (45 min each with no rinsing, 15 passes through each chamber). The higher fluorescence of Bio-`agAtcWcaWgt` compared to Bio-`agTtcWcaWgt` (ratio = 1.8), Bio-

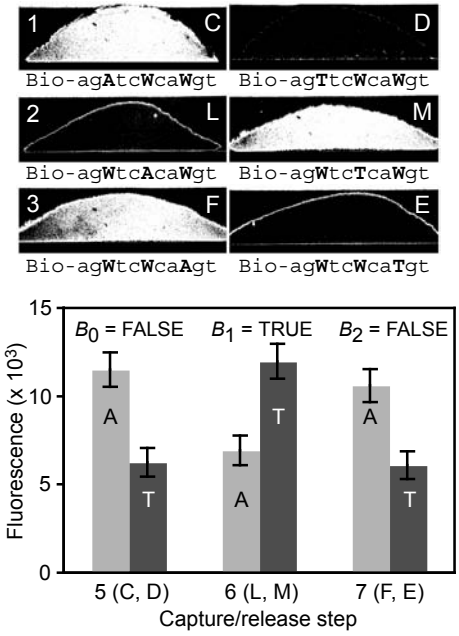


Figure 3.6: Fluorescence images (3 s exposures) of beads in the three pairs of readout steps, with 45 min capture and no rinse for each step. In the analysis of the readout steps, the greater fluorescence intensities in chambers C (Bio-`agAtcWcaWgt`), M (Bio-`agWtcTcaWgt`), and F (Bio-`agWtcWcaAgt`) indicate that the remaining input oligonucleotides are predominantly FAM-acTtgAgaTct. The corresponding values of bits B_0 (FALSE), B_1 (TRUE), and B_2 (FALSE) encode the single correct answer to the problem. Bead fluorescence ratios are 1.8 for step 1 and 1.7 for steps 2 and 3.

agWtcTcaWgt compared to Bio-agWtcAcaWgt (1.7), and Bio-agWtcWcaAgt compared to Bio-agWtcWcaTgt (1.7) indicate that the remaining input DNA was predominantly FAM-acTtgAgaTct, and the corresponding bit values B_0 = FALSE, B_1 = TRUE, and B_2 = FALSE are the correct solution to the problem.

3.4 Discussion

The microfluidic processor presented here enables the fluidic manipulations necessary to utilize single nucleotide polymorphisms in DNA computing. Integrated monolithic membrane valves and pumps efficiently route a μL-scale loop of input oligonucleotide solution selectively between capture chambers. While existing DNA computers typically utilize single-passes of input oligonucleotides through a capture medium [29,50], our microfluidic processor continuously recirculates the input oligonucleotide solution through the capture bead suspension at a rate of nearly one cycle per minute. The resulting improvement in hybridization kinetics makes possible the use of SNPs to represent binary bits in a hybridization-based DNA computation. This novel format requires only 1/8 of the capture time used in a current state-of-the-art computation employing constant 15-base capture sequences to represent each bit [29]. The use of SNPs to encode bits vastly simplifies the preparation of the DNA used in our computation, with a single synthesis run generating all possible answers to the problem. In our SNP computer, single base mismatches are identified with a 1.1% error, and the step-to-step transfer efficiency of 92% during the computation exceeds

the 87% efficiency observed in previous computations [29].

The most significant contribution to error proved to be nonspecific interactions between input DNA and beads early in the computation, which occur when the overall concentration of input DNA is highest. Increasing the rinse time from three to eight minutes for the first computation step reduced the effect of this error. We also found that the additional chambers in the fluid loop during the readout steps decreased the number of cycles of the fluid through each bead suspension from 22 to 15 passes and reduced bit fidelity; increasing the interrogation capture/release time further should compensate for the reduced number of cycles and improve upon the current mean ratio of 1.8 in Figure 3.6. Finally, the use of fewer beads in the readout steps would further increase the fluorescence signal intensity, the sensitivity of the device, and the number of possible processor cycles.

A potential source of error as more complex problems are solved originates from the use of SNPs to represent binary bits. While the use of single nucleotide polymorphism wobbles to encode bits simplifies the task of preparing the input population, the method relies upon the destabilization caused by single-base mismatches for discriminating right answers from wrong ones. Chemical [114] or enzymatic [115] mismatch cleavage agents included in the rinse buffer could be used to nick imperfectly-matched input DNA for more effective elimination during rinsing. Also, since the outcome of the computation depends only upon detecting an excess of one base at each polymorphic locus, a technique like polymorphism ratio sequencing [116] could be used

to detect as little as a 5% excess of one base at each polymorphic site.

The sensitivity of the microfluidic processor to single base mismatches also makes the device well-suited for the analysis of SNPs in biological assays. Assessing risk factors often requires not only the genotype of an individual but also the haplotype [117]. By modeling the haplotyping assay after the computation performed here and using genomic DNA fragments as the input information, it should be possible to obtain correlated SNP information using our microfluidic processor. While the SNPs detected here are separated by only a few bases, the single-base mismatch sensitivity of our device does not rely upon this proximity. Rather, 10- to 15-base capture sequences complementary to distinct and distant polymorphism-containing regions could be used to capture and route predetermined alleles; haplotype information would be revealed by sorting SNP populations through a series of steps similar to the AND and OR operations demonstrated here.

What are the ultimate computational capabilities of our microfluidic processor? Single nucleotide mismatches that reduce the duplex T_m by as little as 4 °C can be detected by hybridization [118]. Using the same experimental conditions as in this work, a nearest-neighbor hybridization thermodynamics model [113] predicts that an 8-bit, 10-clause (10 step) satisfiability problem could be solved using the 10-base input population 5'-FAM-gW_8g-3' with > 4 °C difference between the least-stable perfectly-matched duplex and the most-stable single-mismatch duplex. Using 18 pmol of input DNA and a 92% step-to-step transfer efficiency as demonstrated here, 31 fmol of the

single correct answer would be found at the end of the computation. In a device volume of 5.8 μL, the answer would be present at a concentration of 5.3 nM—an order of magnitude above the limit of detection of the fluorescence microscope used in this study. Such a result would be a significant advance over previous molecular computations employing eight polymorphic bases to solve 4-bit, 5-clause satisfiability problems [119, 120].

Solving problems larger than 8 bits and 10 clauses will require a larger-scale processor, a more extensible encoding scheme, and a more sensitive detection method. We expect that further miniaturization of the processor features (capture chambers, valves, etc.) will be possible and will result in improved capture efficiency, decreased reagent consumption, and reduced computation time. Additionally, since the current 16-chamber device utilizes only half of a 10 cm glass wafer, a processor fabricated on the full wafer could easily be extended to 32 capture/release chambers and used to solve 32-clause problems. A 32-chamber device would still require only two external magnet/heater assemblies, one on each side of the device. The most complex DNA computation completed to date, a 24-clause satisfiability problem, used long (300 base) input DNA containing constant capture sequences of 15 bases to represent each of 20 binary bits [29]. Larger computations can be performed using constant capture sequences because different 15-base sequences can be distinguished more easily than single-base mismatches. We anticipate that the improved capture and transfer kinetics that make single-base mismatch detection possible in our device will also

improve the efficiency of multi-base capture. Using a 25-chamber device and 15-base sequences to represent bits, a state-of-the-art 20-bit, 24-clause (24 step) satisfiability problem solved with 18 pmol input DNA and 92% transfer efficiency would result in 10^6 molecules encoding the single correct answer at the end of the computation—a population easily detectible by PCR and possibly also detectible by direct fluorescence means. We expect that the improved capture kinetics, step-to-step transfer efficiency and automation of our microfluidic processor will make it a valuable platform for many types of next-generation molecular computations.

Acknowledgments

Device fabrication was performed in the University of California, Berkeley Microfabrication Laboratory. We thank Robert Blazej for stimulating discussions. This research was supported by generous donations from Affymetrix to the Berkeley Center for Analytical Biotechnology.

Chapter 4

Development and Multiplexed Control of Latching Pneumatic Valves Using Microfluidic Logical Structures

Reproduced by permission of The Royal Society of Chemistry from "Development and Multiplexed Control of Latching Pneumatic Valves Using Microfluidic Logical Structures" by William H. Grover, Robin H. C. Ivester, Erik C. Jensen, and Richard A. Mathies, *Lab on a Chip* **6**, 2006 (623–631).

Novel latching microfluidic valve structures are developed, characterized, and controlled independently using an on-chip pneumatic demultiplexer. These structures are based on pneumatic monolithic membrane valves and depend upon their normally-closed nature. Latching valves consisting of both three- and four-valve circuits are demonstrated. Vacuum or pressure pulses as short as 120 ms are adequate to hold these latching valves open or closed for several minutes. In addition, an on-chip demultiplexer is demonstrated that requires only n pneumatic inputs to control $2^{(n-1)}$ independent latching valves. These structures can reduce the size, power consumption, and cost of microfluidic analysis devices by decreasing the number of off-chip controllers. Since these valve assemblies can form the standard logic gates familiar in electronic circuit design, they should be useful in developing complex pneumatic circuits.

4.1 Introduction

Microfluidic analysis devices have evolved rapidly from the early single-channel structures [58] to complex microdevices that perform hundreds of assays in parallel [121] and that integrate multiple sample preparation and analysis operations [109]. Microfluidic valves and pumps play a critical role in many of these more complex lab-on-a-chip devices, and valving technologies that provide dense fabrication and parallel pneumatic actuation of hundreds of valves [84, 107] have been very useful in this development. However, these technologies are generally not well suited for *independent* control of large numbers of valves because each independent valve structure requires a dedicated off-chip controller. The size, power consumption, and cost of these off-chip controllers impose a practical limit on the number of independent pneumatic valve and pump operations that can be integrated into a lab-on-a-chip device.

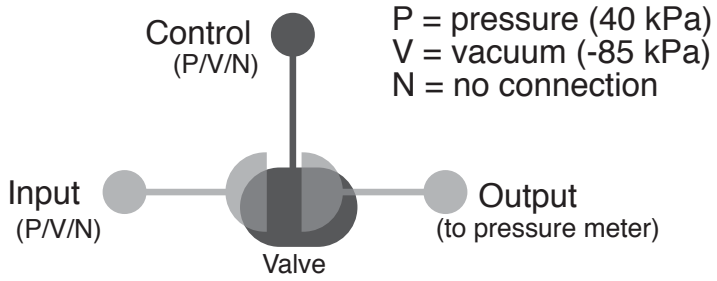
If the valve control circuitry for a microfluidic device could be integrated on-chip using pneumatic networks, then complex control tasks could be performed using only a small number of off-chip controllers. In these digital pneumatic networks or circuits, pneumatic valves would control pulses of pressure or vacuum that actuate other valves. The program to be executed on the microfluidic device is encoded in the layout of the pneumatic valves, and the "output" is the actuation of working (fluidic) valves in the desired on-chip assay. This technology would enable both simultaneous actuation of multiple valves in parallel *and* independent actuation of multiple valves in series. While innovative pneumatic logical structures have been demonstrated

using normally-open soft lithographic valves to address arrays of microreactors and channels [122,123], no on-chip pneumatic system capable of arbitrary addressing and control of dense arrays of independent valves has been reported.

Latching microvalves are critical for the development of an integrated pneumatic addressing system. Since latching valves maintain their open or closed state while disconnected from a controller, a large number of independent latching valves can share a single control line by the principle of time-division multiplexing [124]. Early latching microvalves used bistable, buckled silicon or polymer membranes and electrostatic or thermopneumatic actuation [125–127]. Latching behavior can be imparted to ordinary valves using off-chip pneumatic and electrostatic manifolds [128]. Electromagnets [129, 130] and phase-changing pistons [131] have also been used in latching valve designs. These technologies represent an important step toward addressable valves for large-scale independent control, but complex fabrication as well as chemical and physical incompatibilities limit their practical use in large-scale lab-on-a-chip devices.

The monolithic membrane valves and pumps developed previously by our group [107] have enabled applications in lab-on-a-chip systems for pathogen and infectious disease detection [109], extraterrestrial amino acid analysis [110], pressure-injected electrophoretic separation [132], dielectrophoretic cell concentration [133], nanoliter-scale Sanger DNA sequencing [134], automated evolution of RNA catalysts [135], and SNP-based DNA computation [136]. The normally-closed nature of such a valve

is illustrated in Figure 4.1. When vacuum is applied to the control channel, the polydimethylsiloxane (PDMS) membrane is pulled into the displacement chamber and fluid is free to flow from the input channel to the output channel. The nature of the glass-PDMS bond makes the valves effective for controlling on-chip flows of gas as well: Table 4.1 presents a "truth table" for the six possible assignments of



Rule	Maintained at Input (kPa)	Maintained at Control (kPa)	Measured at Output (kPa)
PP	40	40	0
PV	40	-85	40
PN	40	0	40
VP	-85	40	0
VV	-85	-85	-83
VN	-85	0	0

Table 4.1: "Truth table" for pneumatic logic.

pressure, vacuum, and "no connection" (atmospheric pressure) to the control and input connections of such a monolithic membrane valve. The "normally closed" nature of the valve keeps the valve sealed when equal pressures are applied to the input and control connections, and no pressure reaches the output (Rule *PP*). Input pressure *is* passed undiminished to the output if vacuum is applied to the control (Rule *PV*). If the

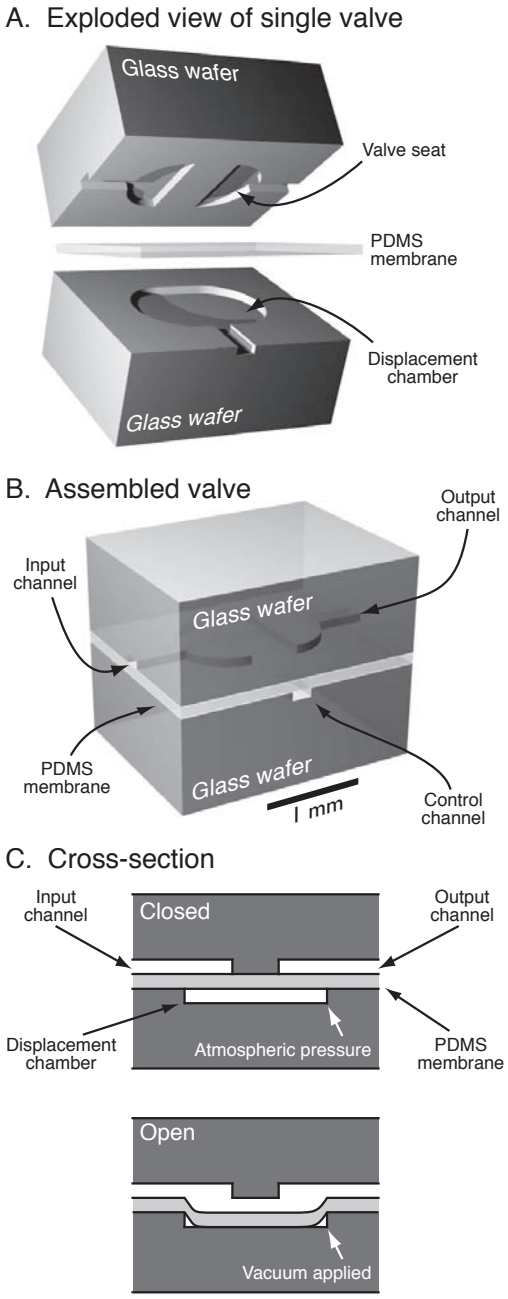


Figure 4.1: Monolithic membrane valve structure before (A) and after (B) bonding the wafers together with the PDMS membrane. Valves consist of a discontinuity in a channel (valve seat) directly across the PDMS membrane from an etched displacement chamber. (C) Lengthwise cross-section through a valve. Valves are normally closed against low pressures in the valved channels; applying a vacuum to the displacement chamber via the "control" channel pulls the membrane into the displacement chamber and opens the valve.

input pressure is large enough to force the valve open, the output can be pressurized even if no connection is made to the control connection (Rule *PN*). Vacuum applied to the input connection seals the valve regardless of whether there is pressure or no connection at the control (Rules *VP* and *VN*). Finally, input vacuum *is* passed to the output if vacuum is applied to the control connection (Rule *VV*), but the valve remains open only as long as the output connection is at a higher pressure than the input and control connections. Once the output vacuum reaches approximately 98% of the input vacuum, the "normally closed" nature of the valve dominates and the valve closes. By applying these rules, valve-based circuits for performing specific on-chip tasks can be designed.

In this work, we present and characterize several useful applications of pneumatic logic structures that exploit the capabilities of our monolithic membrane valves. Simple three- and four-valve networks are developed that function as latching valves. These valves maintain their open or closed state even after all sources of vacuum and pressure are removed from the device. Principles of pneumatic logic are then used to fabricate and test an on-chip valve-based demultiplexer that distributes millisecond duration vacuum and pressure pulses to set the latching valves open and closed. Using pneumatic logical structures, n off-chip pressure/vacuum pneumatic control lines are used to control $2^{(n-1)}$ independent latching valves. These pneumatic logic structures are immediately valuable because they reduce or eliminate off-chip controllers. We anticipate that the operation of complex lab-on-a-chip devices could be programmed

into and controlled by such on-chip pneumatic logical structures.

4.2 Methods

4.2.1 Latching valve device concept

Figure 4.2 depicts a three-valve circuit that forms a vacuum-latching ("V-latching") pneumatic valve. The control for the latching valve is connected to two additional valves, a vacuum valve (responsible for holding the latching valve open by sealing a vacuum on-chip via Rule *VV*) and a pressure valve (responsible for eliminating the sealed on-chip vacuum via Rule *PN*). The resulting circuit holds the valve open or closed after a short vacuum or pressure pulse is applied to the "set pulse input" channels. The related pressure/vacuum-latching ("PV-latching") valve uses trapped vacuum to hold the latching valve open *and* trapped pressure to hold the valve closed against a wider range of fluid pressures.

In the V-latching valve shown in Figures 4.2 and 4.3A, pulses of pressure and vacuum in the "set pulse input" channels are connected to the *input* of a pressure valve and the *input* and *control* of a vacuum valve. Since these valves are actuated by and operate upon pressurized and depressurized air, the usual references to fluidic and pneumatic connections [107] have been discarded in favor of the *input*, *control*, and *output* connections illustrated in Table 4.1. The pressure, vacuum, and latching valves are normally closed (Step 1 in Figure 4.3A). When a pulse of vacuum is applied

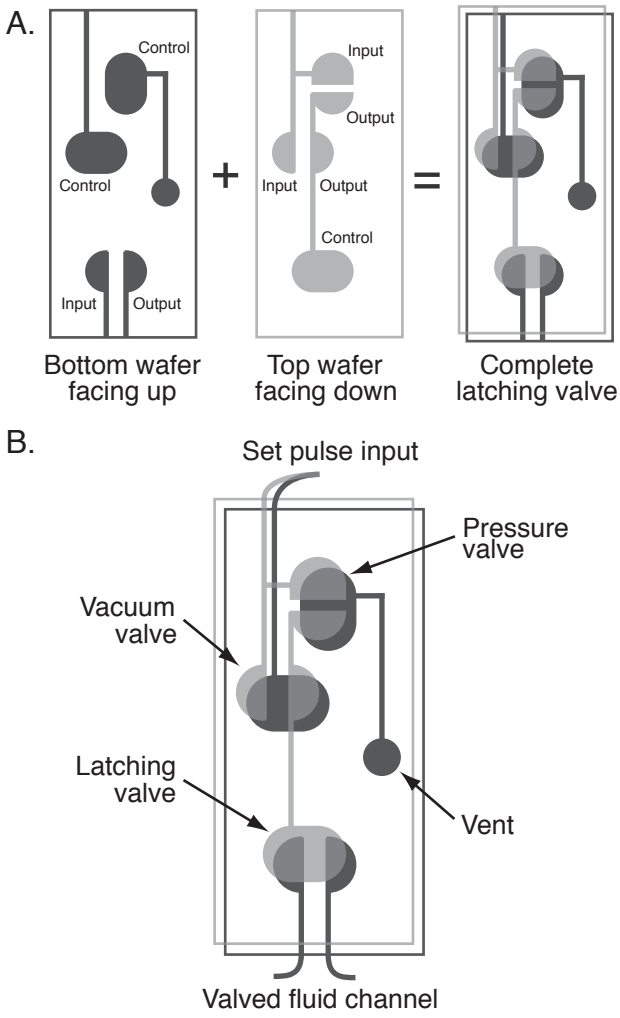


Figure 4.2: (A) Assembly of the latching valve structures. (B) Design of a completed latching valve, including the vacuum and pressure valves that impart latching behavior to the valve.

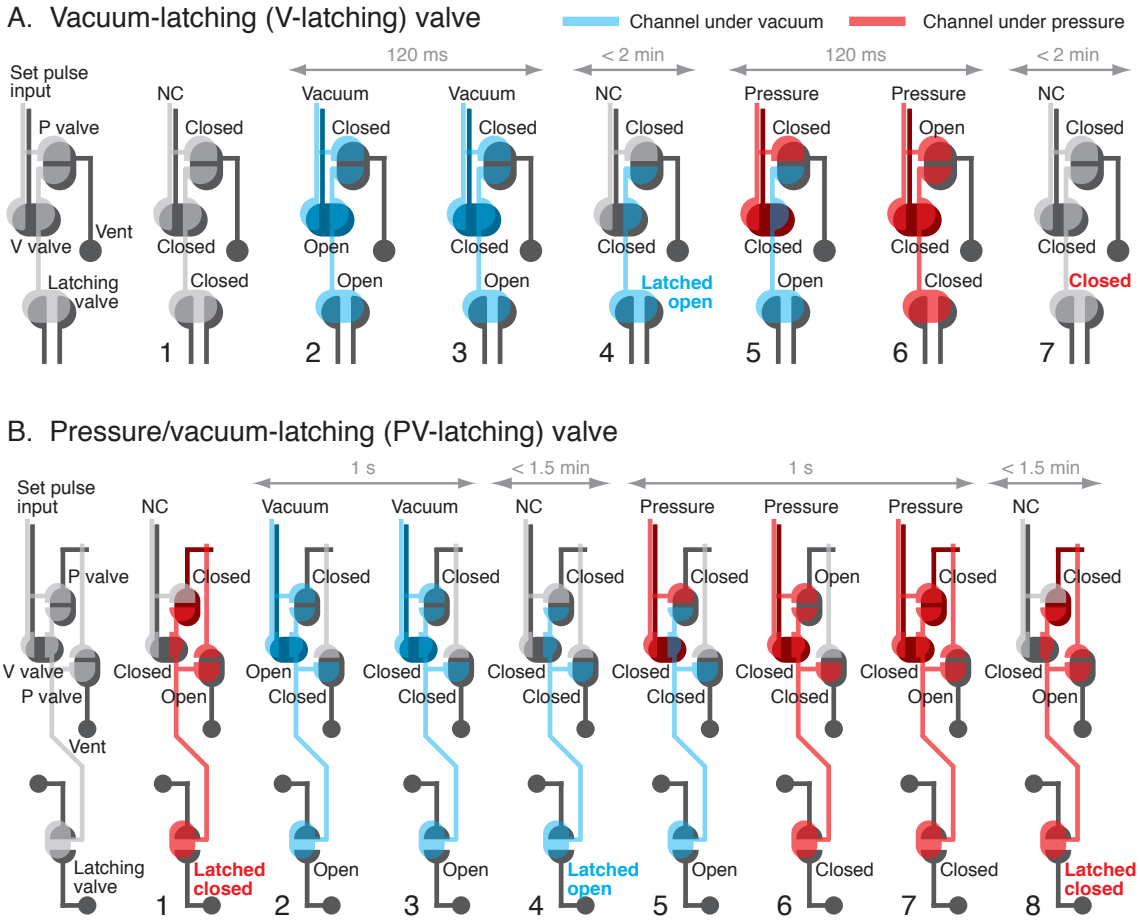


Figure 4.3: Design and operation of the V-latching (A) and PV-latching (B) valve pneumatic circuits. Both circuits contain the actual latching valve and two or three additional pneumatic logic valves. In the V-latching valve, 120 ms vacuum pulses (-85 kPa relative to atmospheric) applied to the "set pulse input" channels depressurize the latching volume and open the latching valve in Step 4. Pressure pulses (120 ms, 40 kPa relative to atmospheric) eliminate the vacuum in the latching volume and close the latching valve in Step 7. "NC" indicates that no connection (only atmospheric pressure) is applied to the "set pulse input" channels. The PV-latching valve opens in a manner similar to the V-latching valve but traps pressure in the latching volume during closure (Step 8); this pressure seals the latching valve closed against fluid pressures as high as 17 kPa. Gray arrows show typical amounts of time for the specified steps.

to the "set pulse input" in Step 2, the vacuum valve opens (Rule *VV* in Table 4.1) and the pressure valve remains closed (Rule *VN*). The *latching volume* (the channel volume containing the outputs of the pressure and vacuum valves and the control of the latching valve) is depressurized, and the latching valve opens. In less than 120 ms, when the latching volume has been depressurized to approximately 98% of the set input vacuum, the vacuum valve closes automatically (Step 3). When the set input vacuum pulse is removed in Step 4, the latching volume is sealed under vacuum by the pressure and vacuum valves according to Rule *VN*, and the latching valve will remain latched open as long as adequate vacuum remains in the latching volume.

To close the V-latching valve in Figure 4.3A, a pulse of pressure is applied to the "set pulse input" in Step 5. Within 120 ms, this pressure forces the pressure valve open according to Rule *PN* in Step 6, and the now-pressurized latching volume seals the latching valve shut. When the set input pressure pulse is removed in Step 7, the pressure in the latching volume escapes as the pressure valve closes. With no pressure in the latching volume to hold it closed, the latching valve can hold off fluid pressures up to 4 kPa without leakage.

The PV-latching valve shown in Figure 4.3B can hold off larger fluid pressures because the latching volume is pressurized while the valve is latched shut (Step 1 in Figure 4.3B). The PV-latching valve is modeled after the V-latching valve but includes a second pressure valve, with its *input* connected to the latching volume, its *output* connected to the control of the first pressure valve, and its *control* connected to

the atmosphere. In Steps 2 through 4, a pulse of vacuum opens the PV-latching valve in a manner similar to the V-latching valve; the second pressure valve remains closed because of Rule *VN*. To close the PV-latching valve, a pressure pulse is applied to the "set pulse input" in Step 5. Within 1 s, this pressure forces open the *first* pressure valve by Rule *PN* in Step 6, then forces open the *second* pressure valve by Rule *PN* in Step 7. With the latching volume and the control for the first pressure valve all pressurized, the first pressure valve closes according to Rule *PP*. When the set input pressure is removed in Step 8, the pressure in the latching volume actively holds the first pressure valve closed and pressure is maintained in the latching volume, thereby holding the PV-latching valve shut against fluid pressures up to 17 kPa without leakage.

4.2.2 Device fabrication

Device features were etched into glass wafers using conventional photolithography and wet chemical etching [111]. Briefly, 1.1 mm thick, 100 mm diameter borosilicate glass wafers were coated with 200 nm of polysilicon using low-pressure chemical vapor deposition. The wafers were then spincoated with positive photoresist, soft-baked, and patterned with the device design using a contact aligner and a chrome mask. After development and removal of irradiated photoresist, the exposed polysilicon regions were removed by etching in SF_6 plasma and the exposed regions of glass were etched isotropically in 49% HF to a depth of 50 μm. After stripping the remaining

photoresist and polysilicon layers, the wafers were diamond drilled with 500 μm diameter holes for pneumatic and fluidic connections. The wafers were then scored and broken, and the resulting layers were bonded together using a 254 μm thick PDMS elastomer membrane [107]. Optionally, two or more etched or drilled glass wafers can be thermally bonded together prior to PDMS bonding; the resulting devices contain all-glass fluid layers that minimize fluid-PDMS contact.

4.2.3 Valve characterization measurements

The latching valve structures were characterized using variable-duration pressure (40 kPa) and vacuum (-85 kPa) pulses from a computer-controlled solenoid valve. Pressures reported are relative to atmospheric pressure and were measured using a strain gauge pressure transducer. Flow rates through latching valves were measured by connecting a variable-height column of water to the input of the latching valve. The valve output was then connected to a short piece of hypodermic tubing suspended in a vial of water on an analytical balance with 1 mg (1 μL) precision. The mass of water flowing through the valve per unit time was used to determine the volumetric rate of flow through the valve and, in turn, the open or closed state of the valve against the applied fluid pressure.

4.3 Results

4.3.1 Latching valve characterization

To test the function of the latching valve structure, fluid flow through a latching valve was measured while pressure and vacuum pulses of varying durations were used to actuate the valve. In the first trace in Figure 4.4A, 60 s of *constant* vacuum or pressure was applied to hold the vacuum-latching valve open or closed. In subsequent traces, shorter *pulses* of vacuum and pressure were used to latch the valve open or closed. The similarity of the traces indicates that latching valves behave identically to constant vacuum/pressure valves, with only 120 ms vacuum/pressure pulses required to reliably actuate the latching valve. Shorter pulses (80 ms) still opened the latching valve reliably but were too brief for reliable closure.

To determine the long-term stability of a valve latched open or closed, flow through a latched valve was measured for ten minutes. The first trace in Figure 4.4B shows that a 120 ms pressure pulse is adequate to latch a V-latching valve closed for at least ten minutes. The second trace indicates that a 120 ms vacuum pulse latches a V-latching valve open for two minutes before the flow rate through the valve decreases by 10%. Due to the gas permeability of the PDMS membrane, a gradual loss of vacuum in the latching volume slowly closes the latching valve and decreases the flow rate further over the next eight minutes.

The pressure/vacuum-latching valve design in Figure 4.3B pressurizes the latch-

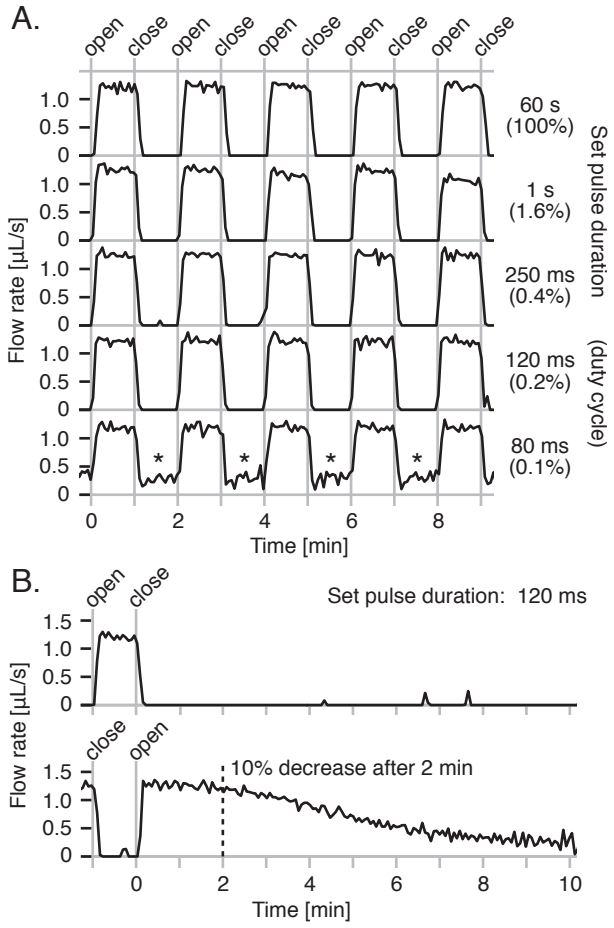


Figure 4.4: (A) Flow rates through a V-latching valve being set open and closed by vacuum and pressure pulses of varying durations. Pulses as short as 120 ms (only 0.2% duty cycle) are adequate for reliable valve actuation; shorter pulses still open the valve but cannot close it against the 4 kPa fluid pressure (asterisks). (B) Flow rates through the same V-latching valve after being latched closed or open by a 120 ms pressure or vacuum pulse. The valve remains closed against the 4 kPa fluid pressure for at least 10 min with only minimal leakage, and open for at least 2 min before the flow rate through the valve decreases by 10%.

ing volume to hold the valve closed against high fluid pressures. To confirm this behavior, V- and PV-latching valves were fabricated with drilled holes for measuring the pressure inside the latching volumes during valve actuation. The pressure inside the latching volume was measured while 10 s pressure and vacuum pulses were used to actuate the valve (Figure 4.5). While both valve designs retained vacuum (-60

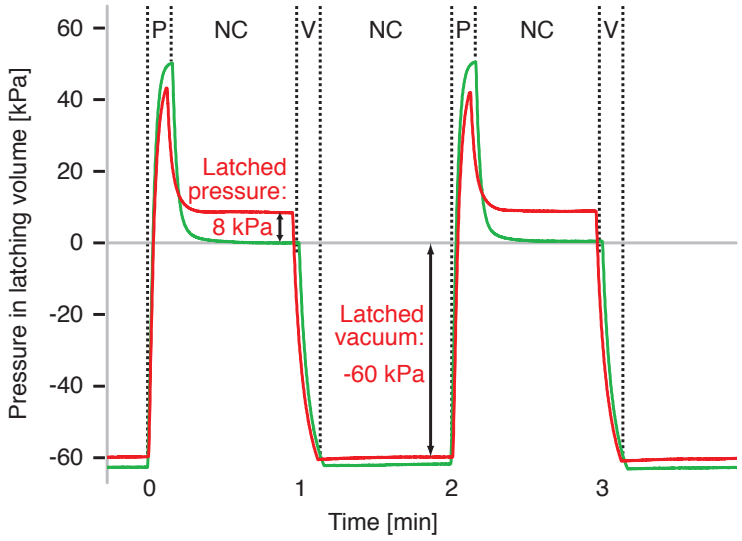


Figure 4.5: Pressure measured inside the latching volume while actuating a V-latching (green) and PV-latching (red) valve with 10 s pressure (P) or vacuum (V) pulses (NC = no connection to valve). While both valves latch open in a similar fashion, only the PV-latching valve traps 8 kPa pressure in the latching volume to hold the valve closed against pressures in the fluidic system.

kPa) in the latching volumes following the vacuum pulse, only the PV-latching valve retained pressure (8 kPa) after the pressure pulse.

To verify that the pressure retained in the PV-latching valve holds the valve closed against high fluid pressures, pressure-driven fluid flow through a PV-latching valve was measured while actuating the valve with 5 s pulses of vacuum and pressure. Figure

4.6A shows that fluid pressures as high as 17 kPa were held off by the valve when latched closed; at 24 kPa, leakage of approximately 1 μL/s was detected through the closed valve. In Figure 4.6B, the shortest pressure pulse required for reliable sealing against 17 kPa fluid pressure was found to be 1 s. This is considerably longer than the 120 ms pulse required to close the V-latching valve, probably because the two pressure valves must open in series via relatively-slow Rule *PN* before the latching volume is pressurized and sealed. Finally, Figure 4.6C confirms that the long-term stability of latched open or closed PV-latching valves compares favorably with V-latching valves. A 5 s pressure pulse seals the PV-latching valve against 17 kPa fluid pressure for 7.5 min before flow through the valve rises to 10% of the open-valve flow rate. The second trace shows that a 5 s vacuum pulse holds the PV-latching valve open for 1.5 min before the flow rate drops by 10%.

4.3.2 Demultiplexer design and operation

The four-bit binary demultiplexer shown in Figure 4.7 addresses 2^4 or sixteen independent V-latching valves and distributes pressure and vacuum pulses to each of them in turn. A single "set pulse input" pressure/vacuum connection at the top of the device in Figure 4.7A provides the pressure and vacuum required to actuate the latching valves. The demultiplexer contains four rows of monolithic membrane valves, with each row containing twice the number of valves of the previous row. Each row of valves in the demultiplexer is controlled by two pneumatic connections to a single

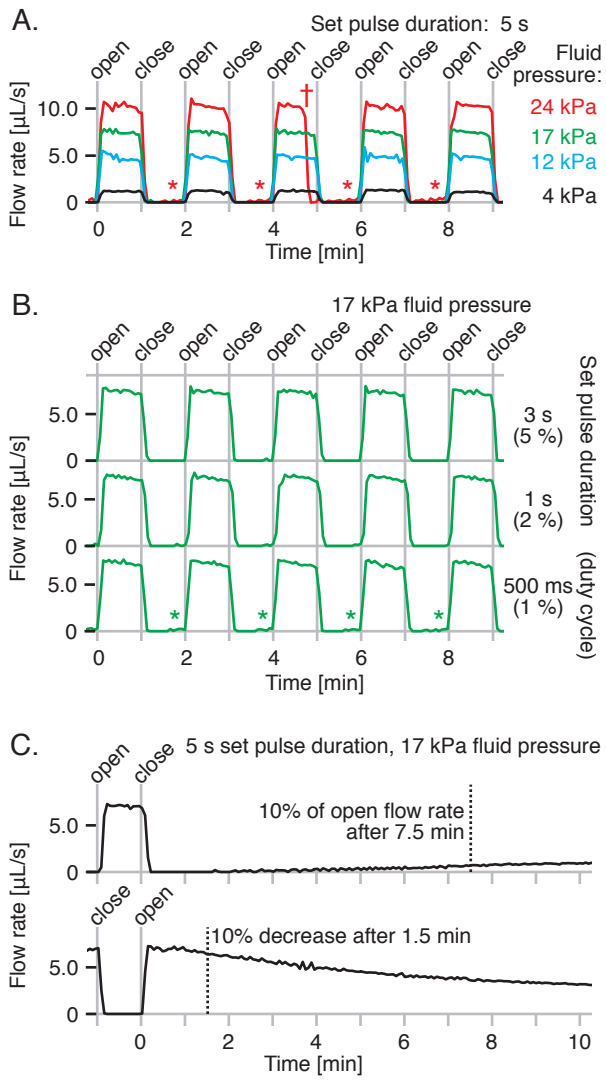


Figure 4.6: (A) Flow rates through a PV-latching valve opening and closing against a range of fluid pressures. The PV-latching valve latches closed against fluid pressures as high as 17 kPa without detectable leakage; higher fluid pressures caused a small amount of leakage through the closed valve (asterisks). A premature valve closure observed only at the highest fluid pressure (dagger) was attributed to residual pressure trapped in the section of the pressure-latching volume between the pressure valves. This pressure leaked into the vacuum-latching volume while the valve was latched open, eliminating the trapped vacuum and closing the latching valve prematurely. (B) Flow rates through the same PV-latching valve, using pressure/vacuum pulses of different durations to open and close the valve. (C) Flow rates through the same PV-latching valve following 5 s pressure or vacuum pulses to hold the valve closed or open against 17 kPa fluid pressure. Leakage through the closed valve increases very slowly to about 10% of the open valve flow rate after 7.5 min.

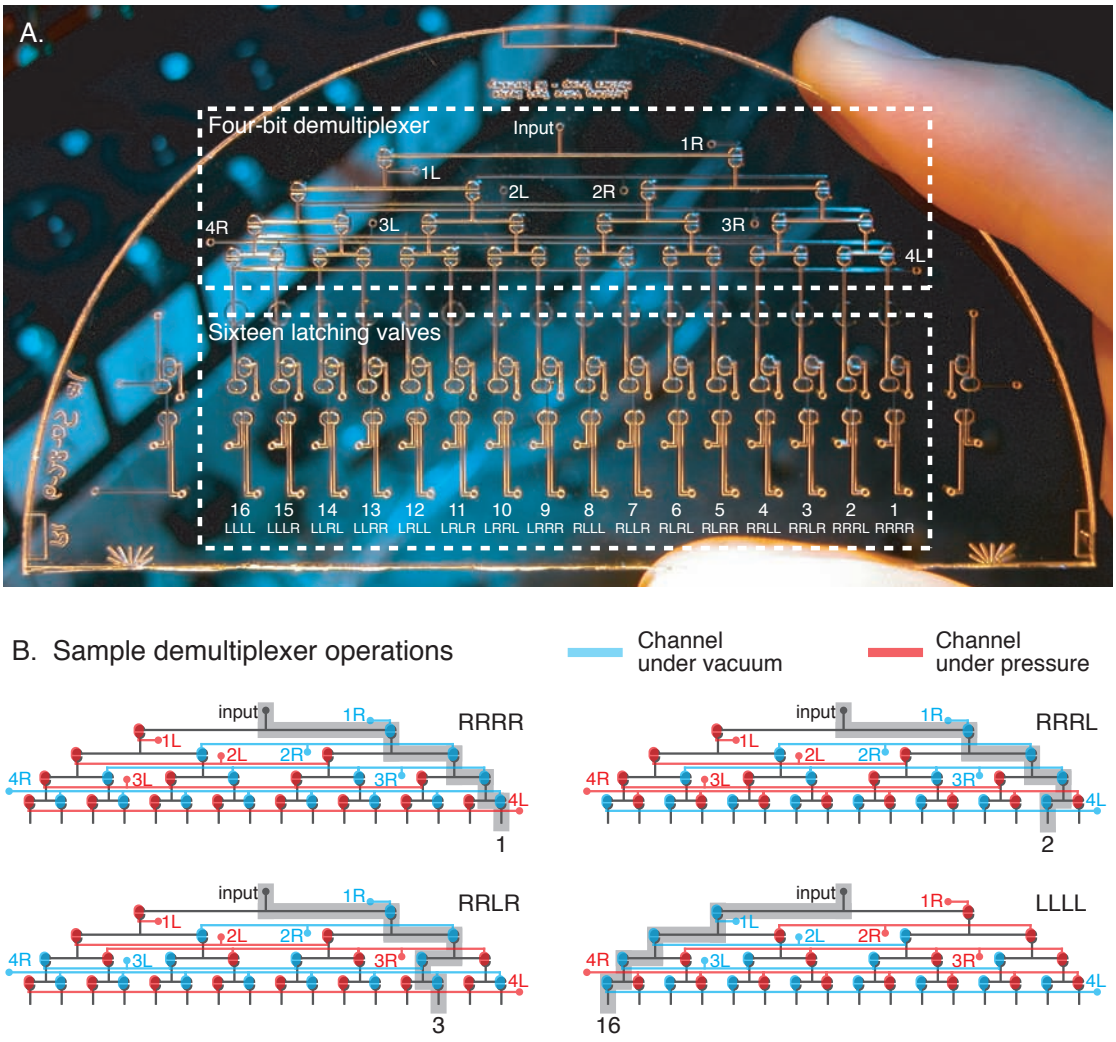


Figure 4.7: (A) Photograph of the multiplexed latching valve test device, with a four-bit demultiplexer (top box) for routing pressure and vacuum pulses from the single "input" connection to each of sixteen latching valves (bottom box). (B) Illustrations of the demultiplexer during four of the sixteen possible addressing operations. Each unique assignment of pressure and vacuum to the four rows of demultiplexer valves routes a single pressure/vacuum source at "input" to one of sixteen latching valves. The device can set a latching valve every 120 ms and set all sixteen valves to any arbitrary pattern every 2 s.

off-chip 4/2 (four connection, two position) solenoid valve. The pneumatic connections are distributed on-chip in an alternating fashion to the demultiplexer valves in each row. For example, in the third demultiplexer row in Figure 4.7, pneumatic connection "3L" controls demultiplexer valves 1, 3, 5, and 7 (numbered left to right), and pneumatic connection "3R" controls demultiplexer valves 2, 4, 6, and 8.

When the solenoid valve controlling a particular row of demultiplexer valves is de-energized, pressure is applied to the odd-numbered demultiplexer valves and vacuum is applied to the even-numbered valves. The even-numbered valves open and "input" pressure or vacuum from the previous row is routed to the *right* into the next row of demultiplexer valves. When the solenoid valve is energized, pressure is applied to the even-numbered demultiplexer valves and vacuum is applied to the odd-numbered valves. The odd-numbered valves open and "input" pressure or vacuum is routed to the *left* into the next row of demultiplexer valves.

An n-bit demultiplexer is addressed by setting each of the n rows to route "input" pressure/vacuum to either the right or the left, and the 2^n possible addresses range from "all right" to "all left" and every intermediate value. For $n = 4$, four of the sixteen possible addresses (RRRR, RRRL, RRLR, and LLLL) are illustrated in Figure 4.7B. Each unique address routes the "input" pressure or vacuum to a different latching valve. By actuating the demultiplexing valves according to a cyclic pattern that selects each latching valve in turn, and applying vacuum or pressure to the "input" connection at the appropriate time to open or close the selected latching

valve, the latching valves can be opened or closed according to any arbitrary pattern. In this manner, an n-row demultiplexer operated by n solenoid valves can control 2^n independent latching valves.

4.3.3 Demultiplexer characterization

A CCD camera was used to record movies of the demultiplexer test device during operation. By cycling the demultiplexer valves through all sixteen addresses in the binary counting order RRRR, RRRL, RRLR, RRLL, RLRR, RLRL, RLLR, RLLL, LRRR, LRRL, LRLR, LRLL, LLRR, LLRL, LLLR, and LLLL, all sixteen latching valves are set in numerical order from 1 through 16 at a rate of 190 ms per step or 3 s per cycle. Figure 4.8 presents a series of movie frames showing the open/closed state of each latching valve at each of the 32 steps in a single demultiplexer cycle. Open valves appear brighter than closed valves because the stretched valve membrane forms a concave surface and reflects additional light from a fiber optic illuminator into the CCD. In steps 1 through 16 vacuum is distributed to open valves 1 through 16 in turn, and in steps 17 through 32 pressure is distributed to close valves 1 through 16. Note that in this and subsequent demultiplexer studies, the state of valve three is intentionally negated, meaning that the demultiplexer must successfully route a single 190 ms pulse of *pressure* (step 4) during a series of 15 *vacuum* pulses, and a single 190 ms pulse of *vacuum* (step 20) during a series of 15 *pressure* pulses—an especially challenging operation for the demultiplexer.

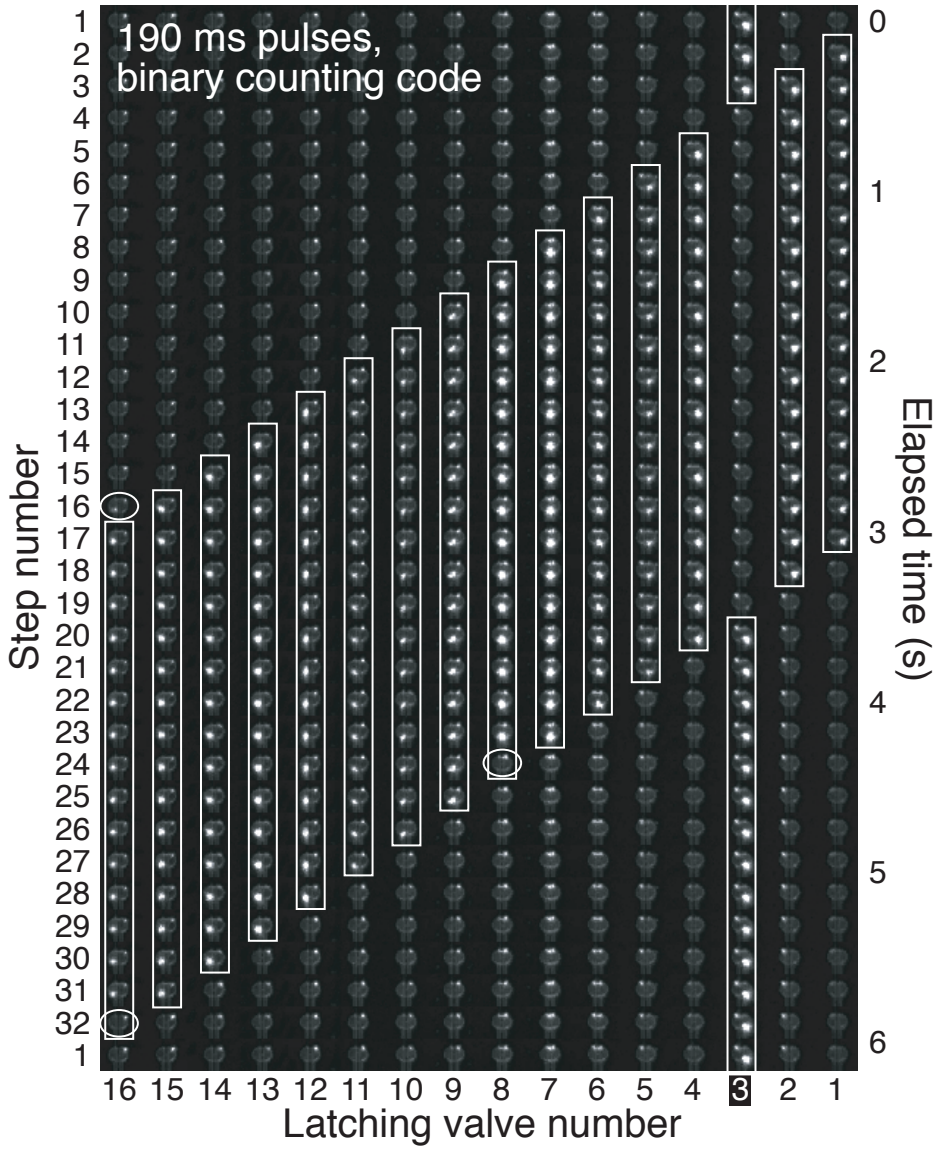


Figure 4.8: Video frames showing the multiplexed latching valve device (Figure 4.7) in operation. The open/closed state of each of the sixteen latching valves is shown at each step in a 32-step cycle; open valves reflect more light and appear brighter than closed valves. A binary counting order was used to address the latching valves through the demultiplexer, with 190 ms pressure/vacuum pulses routed to each of the latching valves in turn. The observed and predicted open valves (white rectangles) at each step agree fairly well, with three errors (white ovals) attributed to demultiplexer malfunctions arising from deficiencies in the binary counting addressing code. The original video is available for download (supplementary information).

While the observed pattern of open valves in Figure 4.8 closely matches the expected pattern (white rectangles), three errors were found (white ovals): valve 8 closed early with valve 7 in step 24, and valve 16 opened early with valve 15 in step 16 and closed early with valve 15 in step 32. Each of these errors involves a valve opening or closing early with the previous valve. Such errors occur when only the least significant bit of the demultiplexer is switching, suggesting a malfunction associated with the least significant row of valves. Closer examination of the binary counting pattern used to operate the demultiplexer revealed that the least significant bit of the demultiplexer switches with *every* step, causing the sixteen demultiplexer valves associated with this bit to open or close every 190 ms. Errors of only a few milliseconds in the actuation of these overwhelmed demultiplexer valves evidently cause the observed errors.

To lessen the repetitive strain on the least significant bit demultiplexer valves, the binary counting order was replaced by the Gray code [137] order RRRR, RRRL, RRLL, RRLR, RLLR, RLLL, RLRL, RLRR, LLRR, LLRL, LLLL, LLLR, LRLR, LRLL, LRRL, and LRRR. This pattern sets the sixteen latching valves in the order 1, 9, 13, 5, 7, 15, 11, 3, 4, 12, 16, 8, 6, 14, 10, and 2 at a rate of only 120 ms/step or less than 2 s per cycle. Using this addressing order, demultiplexer valves are actuated at most every *other* step, or every 240 ms, compared to every 190 ms for the flawed binary counting order. The video frames in Figure 4.9A show the open/closed state of each latching valve at each of the 32 steps in a single demultiplexer cycle that

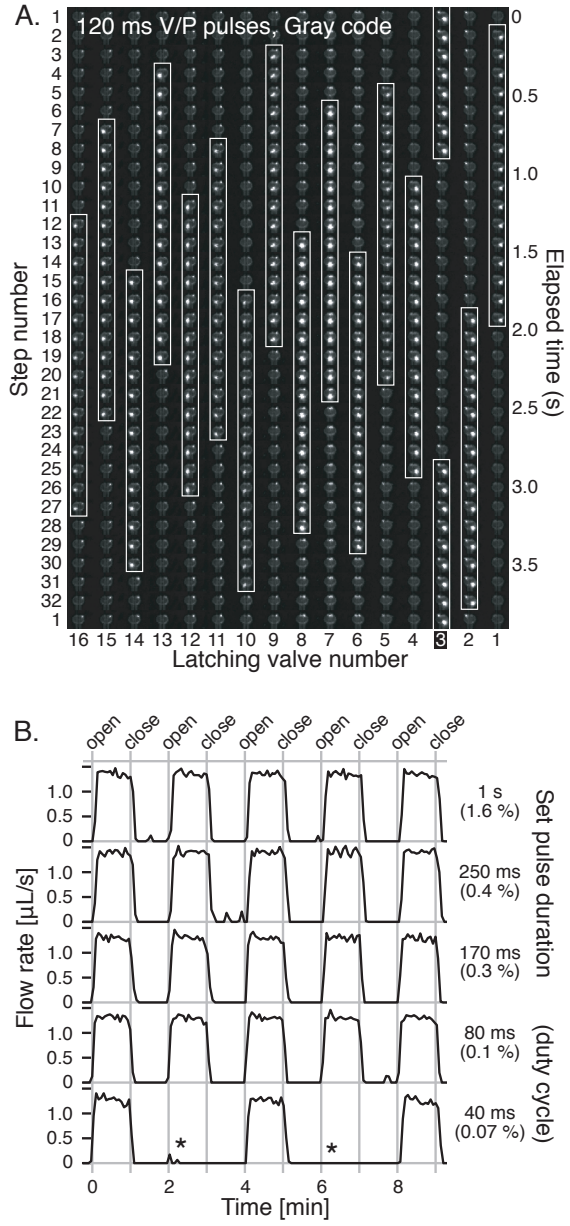


Figure 4.9: (A) Video frames showing the multiplexed latching valve test device in operation, using an improved Gray code order for operating the demultiplexer and only 120 ms pressure/vacuum pulses. The observed open valves matched the expected open valves (white rectangles) with no errors. (B) Flow rates through inverted latching valve 3, obtained while operating all sixteen latching valves according to the complex actuation pattern shown in A. Pressure and vacuum pulses as short as 80 ms (0.1% duty cycle) were adequate to open and close the valve without errors. Missed openings observed at even shorter pulse times (asterisks) are caused by demultiplexer timing errors. The original video is available for download (supplementary information).

opens each valve in steps 1 through 16 and closes each valve in steps 17 through 32 (with valve 3 still inverted). The observed pattern of open valves exactly matches the expected pattern (white rectangles) with no errors, proving that the demultiplexer can accurately route pressure and vacuum pulses as short as 120 ms to the intended latching valves.

In addition to confirming the operation of the demultiplexed latching valves visually, the ability of the demultiplexed valves to control fluid was also demonstrated. Figure 4.9B presents the flow of fluid through the inverted valve 3 while all sixteen latching valves were being actuated according to the complex pattern in Figure 4.9A. Pressure and vacuum pulses as short as 80 ms were adequate to open and close the inverted valve 3. Shorter pulses occasionally failed to open the valve, probably because of demultiplexer timing errors at fast actuation rates.

4.4 Discussion

We have developed and characterized a latching pneumatic valve design suitable for high-density integration into lab-on-a-chip devices. By eliminating the need for a separate off-chip controller for each independent valve or parallel array of valves on-chip, the latching pneumatic valves presented here make large-scale control of independent valves feasible. The vacuum-latching valves can control on-chip fluid flow in a variety of assays involving low (<4 kPa) fluid pressures, and the pressure/vacuum-latching valves close reliably against fluid pressures up to 17 kPa. Latching valves

retain the low (∼10 nL) dead volumes found in monolithic membrane valves [107]. Since the latching structures presented here consist of monolithic membrane valves that can be operated continuously for hours and for tens of thousands of actuations without failure [135, 136], we anticipate that the long-term durability of these structures will be very favorable. Our latching valve structures depend upon the normally-closed nature of membrane valves [107]. Rules *PN* (input pressure breaking through an unpowered valve), *VN* (input vacuum sealing an unpowered valve), and *VV* (a valve opening to evacuate a volume on-chip, then closing automatically to seal the volume under vacuum), all of which are essential to the operation of the latching valves, would be difficult or impossible to replicate using normally-open PDMS valves [84].

We have also presented a valve-based pneumatic demultiplexer for controlling large arrays of independent latching valves. The demultiplexer uses only n off-chip pneumatic inputs to control $2^{(n-1)}$ multiplexed latching valves. In this example, sixteen independent latching valves can be set in any arbitrary pattern every two seconds using only five pneumatic controls. The multiplexed latching valves retain their ability to independently control fluid flow. Since the pressure, vacuum, and demultiplexer valves that operate the latching valves never contact the valved fluid, the potential for cross-contamination between multiplexed latching valves is eliminated. Existing methods of on-chip logic using normally-open valves have proved to be very useful in addressing rectilinear arrays of microreactors [122] but have not been applied to the

arbitrary control of independent latching valves as presented here.

What is the upper limit for large-scale control using the multiplexed latching valves developed here? Vacuum and pressure pulses as short as 120 ms (8 valves per second) were found to be adequate to hold the V-latching valves open and closed for at least two minutes. In two minutes, *1000* independent latching valves can be set at a rate of 8 valves per second. *This massive number of valves would require* $(\log_2 1000)+1$ *or only 11 off-chip pneumatic controls!* The 10-bit demultiplexer would contain $2^{10+1}-2$ or 2046 valves, and each of the 1000 V-latching valves would require two logic valves, for a total of 4046 on-chip logic valves to control 1000 latching valves. If each logic valve and its associated pneumatic channels occupy 2 mm^2, 4000 logic valves could be fabricated using photolithography into a single glass-PDMS-glass layer of a 10 cm diameter microfluidic device. One surface of this layer could then be bonded to additional glass wafers through another PDMS membrane, thereby forming a fluidic layer for the placement of the 1000 independent latching valves in the desired assay configuration. The prospect that a single additional layer in a lab-on-a-chip device could eliminate literally *hundreds* of off-chip solenoid valves, relays, and computers attests to the potential of pneumatic logical structures.

By reducing the off-chip control equipment necessary for the operation of microfluidic devices, multiplexed latching pneumatic valves should play an especially important role in making low-cost, low-power, and handheld lab-on-a-chip analysis devices a reality. Analysis devices with fewer off-chip solenoid valves and electronic

control circuits would consume less power and be better suited for battery-operated field use. Critically, in robotic analysis systems for space exploration [110], eliminating off-chip controllers would conserve sparse payload space and power. We also note that pneumatic logic circuits like the demultiplexer presented here are immune to high energy particles, solar flares, and electromagnetic pulse interference that can irreparably damage electronic logic circuits [138].

The technology presented here also establishes the basis for pneumatic logic *gates*—generic, valve-based AND, OR, and NOT structures that can be arranged into circuits or programs that encode and control the operation of any microfluidic device. In a classic example, flow through two valves connected in series is allowed only if both valves are open—a logical AND. Similarly, flow through two valves connected in parallel is possible if either (or both) of the valves is open—a logical OR. The feedback loops used to hold the latching valve open in the V-latching valve and closed in the PV-latching valve are closely analogous to NAND- and NOR-based latch circuits used as binary memories in electronic circuits [124]. These logical operations form the foundations of all electronic computations. We anticipate that microfluidic logic structures of the type presented here will similarly prove to be fundamentally useful in the assembly of complex pneumatic microprocessors.

Acknowledgments

Device fabrication was performed in the University of California, Berkeley Microfabrication Laboratory. This research was supported by donations from Affymetrix to the Berkeley Center for Analytical Biotechnology.

Chapter 5

Further Developments in Microfluidic Processor Design and Operation

Since their initial demonstration [107], monolithic membrane valves and pumps have found uses in a variety of applications, both computational and otherwise. Tailoring the valves and pumps to specific applications often required going beyond the designs and protocols first presented in 2003. In this chapter, several new developments in microfluidic processor design and operation are discussed.

5.1 Bus valves

The original monolithic membrane valve design is well-suited for controlling fluid flow through straight channels [107] but is less suitable for applications that require routing of fluids through branches in the fluid channels [136]. In particular, branched networks of channels valved using the original valve design will include small regions in the fluid channels that cannot be rinsed effectively. In the fluidic manifold shown in Figure 5.1A, four original valves are arranged in parallel along a bus channel to supply two different reagents (red and green) to the output channel. Red reagent is first delivered to the output in Step 1. When the bus channel is rinsed briefly in Step 2, a small amount of red reagent remains trapped in the dead-end channel leading up to the valve. In Step 3, the green reagent delivered to the output is contaminated by red reagent diffusing out of the unrinsed valve channel and into the bus channel. Note that the same problem affects on-chip distributors that route fluid from a single input into one of several possible destination channels (the reverse of the operation shown in Figure 5.1A). These manifold and distributor structures play a crucial role

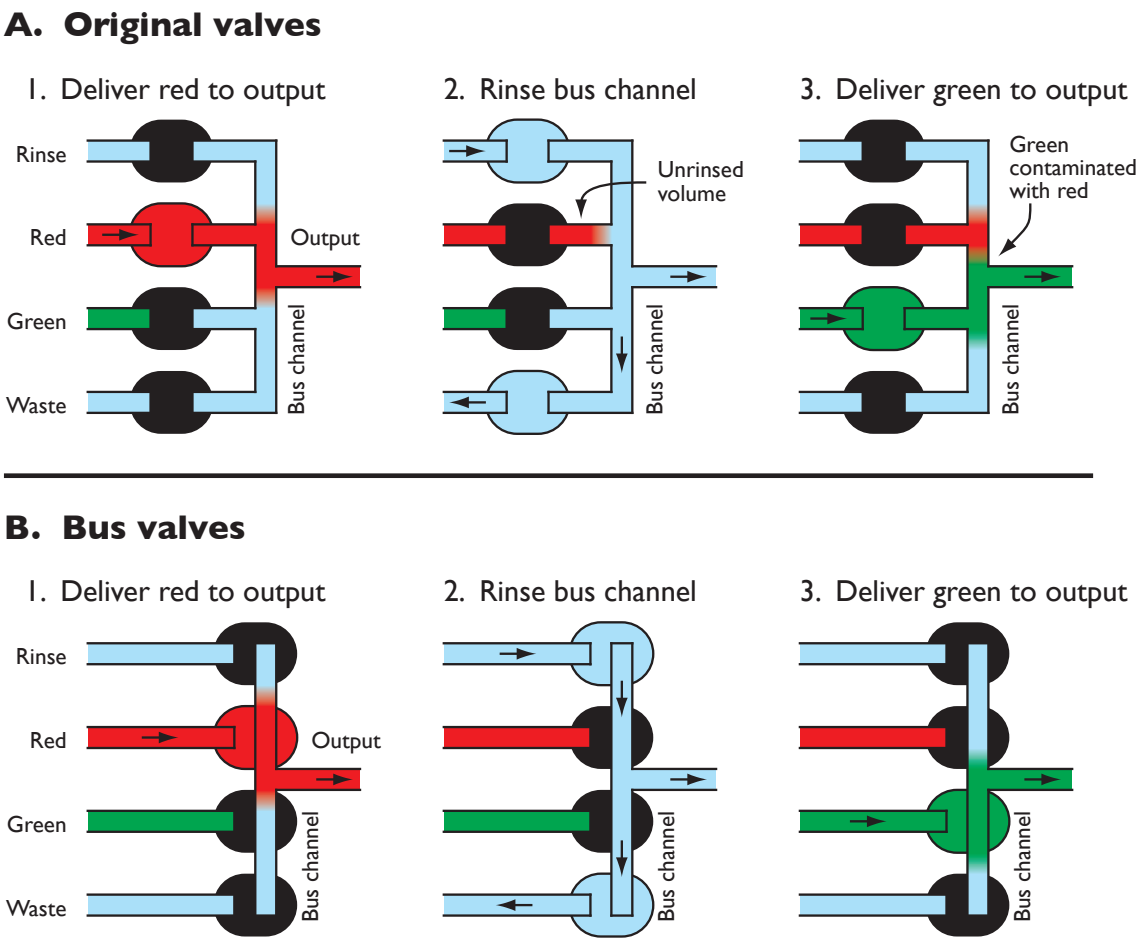


Figure 5.1: Comparison of an on-chip manifold valved using the original valve design (A) and the bus valve design (B). Black valves are closed and arrows indicate the direction of fluid flow. In this example, red and green fluids are selectively routed to the output channel while minimizing red-green cross-contamination within the manifold. In A, the unrinsed red volume in Step 2 diffuses into the bus channel and contaminates the green fluid in Step 3. Bus valves in B eliminate the unrinsed volumes and avoid red-green cross-contamination.

in devices that combine multiple reagents or aliquot a sample on-chip, so a technology for eliminating cross-contamination in these structures would enable their use in many applications that are sensitive to cross-contamination.

The bus valve design shown in Figure 5.1B solves the channel rinsing problem by moving the bus channel *inside* the valve structure. Even though it runs through the four valves, the bus channel is essentially an featureless channel when the bus valves are closed, and can be rinsed from top to bottom as effectively as an ordinary channel. Following delivery of red reagent in Step 1, the bus channel is rinsed in Step 2 until no red reagent remains in the channel. In Step 3, green reagent is delivered uncontaminated to the output channel. Note that if the structure in Figure 5.1B is operated backwards as a distributor, the bus valves still serve to eliminate cross-contamination between samples routed to the different valved channels. Finally, the operations shown in Figure 5.1B eliminate red contamination of the green reagent and vice-versa, but some diffusional contamination of the red and green reagents with the *blue* (rinse) reagent is still possible. By simply "rinsing" the bus channel with red reagent *before* actually supplying red reagent to the output, *all* forms of cross-contamination (green reagent or blue rinse buffer) can be eliminated.

Bus valves have found uses in a variety of devices that require manifolds or distributors to operate upon multiple samples in series. They are used to select which capture chambers will be incorporated into the recirculating loop in the microfluidic processor [136]. Bus valves have also been used to manipulate different samples with

low cross-contamination in instruments for automated amino acid composition and chirality analysis [110], and to introduce reagents into and remove products from rinsable recirculating loops for automated RNA evolution [135].

5.2 Bead-resistant valves

The first successful design for a monolithic membrane valve included small *valve seat channels*—extensions of the fluid channel into the area of the valve across the elastomer membrane from the displacement chamber [107]. As discussed in Section 2.2.2, some extension of the fluid channel into the valve is necessary to decrease the pressure differential required to open the valve. The extended fluid channel provides additional surface area that gives the applied vacuum leverage to pull the membrane away from the fluid wafer when opening the valve. These valve seat channels were intentionally kept small to reduce the amount of dead volume that they contribute to the valve.

Valves based on this original design are unsuitable for applications involving on-chip suspensions of beads [136]. Figure 5.2A shows a photograph of a pump valve after one hour of pumping a 1% suspension of 1 μm diameter magnetic beads and several minutes of rinsing with bead-free buffer. All surfaces of contact between the PDMS membrane and the fluid wafer are contaminated with trapped beads. The most significant bead contamination actually lies outside of the area contained within the valve, indicating that the membrane is separating from the fluid wafer outside of

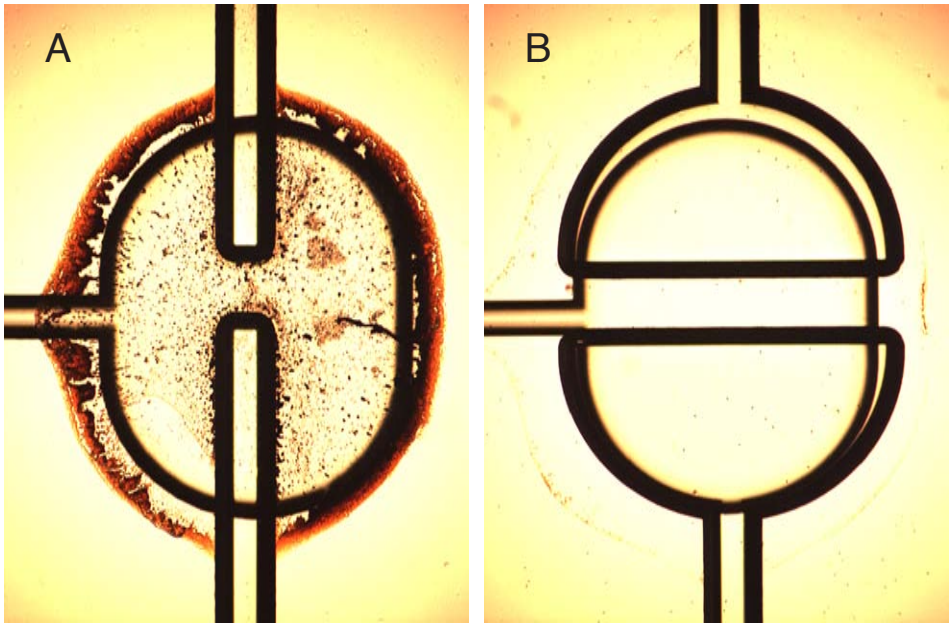


Figure 5.2: Photographs of original (A) and bead-resistant (B) valves after pumping a bead suspension. Valves are 1.0 mm long, 1.5 mm wide, and etched 100 μm deep.

the region of the displacement chamber where vacuum is applied. This separation—a result of the *thinning* of the PDMS membrane as it stretches into the evacuated displacement chamber—provides a shallow trap for micron-sized particles like beads. As the valve closes, beads held in the valve are pressed against the unetched surface of the fluid wafer. If the trapped beads begin to interfere with the sealing of the PDMS membrane against the valve seat, the valve will no longer close completely and the pump will be rendered inoperable. Beads trapped in this manner are very resistant to removal by rinsing, and devices contaminated with beads usually have to be disassembled, cleaned, and re-assembled using a fresh PDMS membrane.

The design of the bead-resistant valve shown in Figure 5.2B reduces bead entrap-

ment by reducing the area of contact between the PDMS membrane and the unetched fluid wafer surface. By expanding the valve seat channels in the fluid wafer to fill most of the valve, very little glass contacts the PDMS membrane when the valve closes. Increasing the size of the valve seat channels also significantly decreases the pressure differential required to open the valve [107]. In addition, valves with larger valve seat channels tend to have longer shelf-lives than valves with smaller valve seats, which can can become stuck shut after several weeks of inactivity. The small amount of beads trapped around the edge of the bead-resistant valve in Figure 5.2B could be eliminated by increasing the size of the valve seat channels another 100 to 200 μm. While bead-resistant valves do have larger dead volumes than ordinary monolithic membrane valves, the additional few nanoliters should be inconsequential for most applications.

Bead resistant valves have been included in several devices that require reduced susceptibility to bead entrapment or more reliable actuation. The valves are used as pump valves in the microfluidic processor to avoid bead loss in the pump and possible cross-contamination of the different beads pumped into the processor [136]. They have also been used in devices like the Sanger sequencing bioprocessor [134], which benefits from the increased long-term reliability of the bead-resistant valves even though no beads are used in the device. In devices that manipulate fluids containing cells (bacteria, red blood cells, etc.), the bead-resistant valve is also expected to be resistant to cell entrapment or lysis during operation. Finally, the large contact area

between the pneumatic and fluidic layers in the bead-resistant valve should make the structure effective at degassing fluid passing through the valve. By applying constant vacuum to the bead-resistant valve, bubbles and dissolved gases could be pulled across the gas-permeable PDMS membrane and eliminated from fluid in the valve. Such an on-chip degasser could reduce or eliminate bubble formation in devices that perform on-chip heating or thermal cycling [109].

5.3 Alternative pneumatic fluids

The gas permeability of PDMS is both a blessing and a curse in monolithic membrane valve devices. While bubbles in the fluid channel can be eliminated by applying a vacuum to the pneumatic channel across the membrane from the bubble, bubbles can also be formed in the fluid channel if too much air pressure is applied to the pneumatic channel. Although air pressures around 10 kPa are generally adequate to seal valves without introducing bubbles [136], applications involving on-chip heating (like thermal cycling) typically require higher air pressures to keep the valves sealed shut.

The formation of bubbles caused by high air pressure can be eliminated if a *liquid* pneumatic fluid is used to actuate the valve. Figure 5.3 presents a series of images of a glycerol-actuated valve filling (Steps 1 through 5) and operating (Step 5 and 6). Initial tests using water as the pneumatic fluid had mixed results: while the pneumatic channels filled with water successfully, applying a vacuum to the water to open the

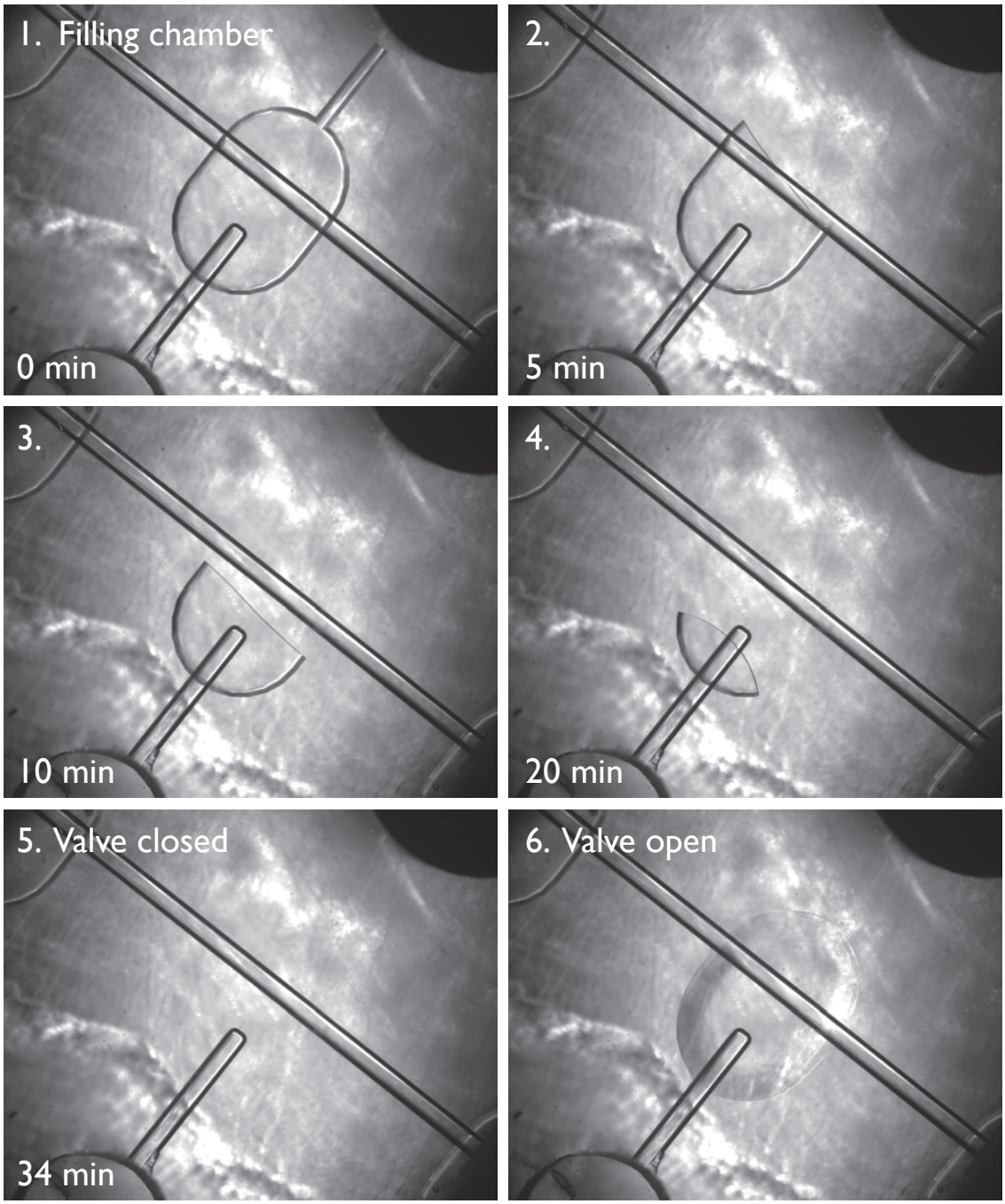


Figure 5.3: Forcing glycerol into the displacement chamber of a valve (1 through 5) and actuating the glycerol valve (5 and 6).

valve resulted in the formation of bubbles in the pneumatic channels from dissolved gases in the water. Glycerol was chosen as an alternative to water because of its low vapor pressure (less than 0.01 kPa at 25 °C compared to 3.2 kPa for water) and its decreased capacity for dissolved gases [139]. The high boiling point of glycerol (290 °C) makes it suitable for use in microfluidic devices at temperatures approaching the boiling point of water. Finally, the index of refraction of glycerol ($n = 1.474$) almost exactly equals the index of refraction of the glass used to fabricate the microfluidic device ($n = 1.472$; Borofloat borosilicate glass, Precision Glass and Optics, Iserlohn, Germany). As a result, the pneumatic channels become invisible after filling with glycerol.

The glycerol-actuated valve in Figure 5.3 is filled with pneumatic fluid in Steps 1 through 5 by applying a pressurized reservoir of glycerol (40 kPa) to the drilled pneumatic input for the valve. The air trapped inside the pneumatic channel is slowly forced across the membrane and into the (empty) fluid channel, and after 34 minutes the displacement chamber of the valve is completely filled with glycerol (Step 5). Only the fluid channels of the closed valve are visible in Step 5. When vacuum is applied behind the glycerol reservoir in Step 6, a few hundred nanoliters of glycerol is removed from the displacement chamber and the PDMS membrane rises into the chamber, opening the valve. With a much lower index of refraction that either the glass or the PDMS, the air in the "fluid" channel of the open valve is visible as a shallow blister in Step 6.

Glycerol-actuated valves open and close against fluid flow in a manner identical to gas-actuated valves, only much more slowly. While gas-actuated valves can be opened and closed many times per second [107], identical glycerol-actuated valves require around 10 seconds to open (-80 kPa vacuum) and around 20 seconds to close (40 kPa pressure). This reflects the finite rate at which a viscous liquid like glycerol can be pulled out of or pushed into the valve's displacement chamber. A less-viscous liquid like mineral oil could be pushed into and out of the displacement chamber much more quickly, resulting in a faster liquid-actuated valve. Finally, this demonstration of liquid-actuated pneumatic valves opens up the possibility of eliminating air pressure and vacuum sources entirely in favor of small pistons or other actuators that pressurize and depressurize the pneumatic fluid directly, a proposal that is discussed in greater detail in the Prospects chapter.

5.4 Pump actuation cycles

Monolithic membrane diaphragm pumps can be operated according to several different *actuation cycles*—the order in which the pump's valves are actuated during pumping. These cycles differ in the pulsatile nature of their flow, their efficiency, and their suitability for different device designs. By comparing different actuation cycles, optimum actuation cycles for a given device or application can be selected.

All pump actuation cycles are governed by two rules about the operation of monolithic membrane valves. The first rule describes the opening of valves on-chip:

Rule 1 *A valve will open only if there is an unobstructed supply of fluid to be pulled into the opening valve.*

The "unobstructed supply of fluid" can come from a drilled reservoir through a channel (possibly containing open valves), or from the *closing* of another valve. A second rule describes the closing of valves on-chip:

Rule 2 *A valve will close only if there is an unobstructed destination for fluid pushed out of the closing valve.*

Again, the "unobstructed destination for fluid" can be a drilled reservoir connected to the valve by a channel (perhaps containing open valves), or an *opening* valve. These rules are a result of the incompressibility of fluid in the microfluidic device and do not apply to valves controlling flows of compressible gases, like the vacuum and pressure valves that operate latching valves [140]. Also, pumps operating within *sealed* sections of microfluidic devices (regions with no open connections to drilled reservoirs, as in [107] and [135]) are subject to additional restrictions discussed in Section 5.6.

For a valve to open *completely*, the supply of fluid being pulled into the valve must be greater than or equal to the fluid capacity of the open valve. Conversely, for a valve to close completely, the capacity of the destination for fluid being pushed out of the closing valve must be greater than or equal to the fluid capacity of the open valve. These observations are of little importance for valves pulling fluid from or pushing fluid to drilled reservoirs because such reservoirs typically contain one or

two orders of magnitude more fluid than an open valve. However, these observations are critically important for valves pulling fluid from or pushing fluid to other valves, as is the case when valves are combined to form pumps. An opening valve pulling fluid from a smaller closing valve will only open partially, and a closing valve pushing fluid to a smaller opening valve will not close.

In this analysis of pumping cycles, small volumes of fluid are followed from valve to valve as a pump actuates. In the simplest case, a pump consists of three valves—called *Input*, *Diaphragm*, and *Output* valves—connected in series and controlled independently [107]. When open, these three valves contain volumes of fluid V_I, V_D, and V_O, respectively. In the following tables, the arrows $\xrightarrow{V_I}$, $\xrightarrow{V_D}$, and $\xrightarrow{V_O}$ represent the flow of the contents of an Input, Diaphragm, or Output valve to the *right* (in the *forward* or intended direction of pumping), and the arrows $\xleftarrow{V_I}$, $\xleftarrow{V_D}$, and $\xleftarrow{V_O}$ represent the flow of an Input, Diaphragm, or Output valve's volume to the *left* (in the *backward* or unintended direction of pumping. The *efficiency* ϵ of a pumping cycle is the fraction of the total number of steps per cycle that is spent actually pumping fluid in the forward direction. Finally, the pulsatility p of a pumping cycle is the fraction of steps per cycle spent pumping fluid either forward *or backward*. By comparing pumping cycles in these generic terms, cycles optimized for particular devices or operations can be identified and used.

The first pumping cycle developed for the monolithic membrane valves was the five-step cycle [107] shown in Table 5.1. The five-step cycle begins in step 1 with

Step		Input		Diaphragm		Output	
1	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_O}$
2	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
3	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
4		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
5		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
Net:	$\xrightarrow{V_D}$		$\xrightarrow{V_D}$		$\xrightarrow{V_D}$		$\xrightarrow{V_D}$

Table 5.1: A single iteration of the five-step actuation cycle for a three-valve diaphragm pump.

the Input open and the Diaphragm and Output closed. In step 2, fluid is pulled forward through the Input as the Diaphragm opens. The contents of the Input are then pushed *backwards* as the valve closes in step 3. When the Output then opens in step 4, fluid flows *backwards* into the valve. In step 5, the Diaphragm closes and pumps its contents forward through the Output. Finally, the Output closes and the Input opens as the cycle repeats and returns to step 1.

As shown in Table 5.1, the net effect of the five-step cycle is the pumping of one diaphragm valve's volume forward per cycle. The five-step cycle has a fairly low efficiency of $\epsilon = 1/5$ or 20% because the volumes V_I and V_O pumped forward in step 1 are balanced by *backward* pumping of the same volumes in steps 3 and 4. The cycle also has a high pulsatility of $p = 3/5$ or 60% because three of the five steps push or pull volumes of fluid to or from the downstream (forward) direction. In practice, this means that features downstream from the pump (cells, beads, etc.) will experience three pulses of fluid flow (two forward and one backward) per five-step actuation cycle. This "two steps forward, one step back" behavior can create bubbles

while loading dry devices with fluid. Channels filled during the "two steps forward" are suddenly emptied during the "one step back." The remaining fluid along the channel walls can coalesce into a plug and trap an air bubble in the channel. If this occurs in a wide region of channel (a reactor, capture chamber, or drilled via hole), the bubble will likely remain in the wide channel and may expand if the device is subsequently heated. For this reason, the five-step cycle is unsuitable for applications requiring the bubble-free transfer of fluid through an initially-empty channel. Finally, the cycle *is* suitable for use with pumps containing diaphragm valves that are larger than the input and output valves. In this case, steps 2 and 5 (opening and closing the diaphragm valve) require more time than steps 1, 4, and 5. In practice, since the input and output valves make no contribution to the volume pumped per actuation in the five-step cycle, pumps using this cycle should have small input and output valves flanking a diaphragm valve sized to deliver the desired volume per actuation [107].

The five-step pump actuation cycle is actually an abbreviated version of the six-step cycle [136] shown in Table 5.2. Step 1 from the five-step cycle is split into two steps in the six-step cycle; the remaining steps are unchanged. Like the five-step cycle, the six-step cycle pumps one diaphragm valve's volume forward per cycle. The additional step decreases the efficiency of the six-step cycle to only $\epsilon = 1/6$ or 17%. The pulsatility of the cycle decreases to $p = 3/6$ or 50%, and inspection of Table 5.2 indicates that the six-step cycle subjects downstream features to at most *two* pulses in series (steps 5 and 6) compared to *three* pulses in series (steps 4, 5,

Step		Input		Diaphragm		Output	
1	$\xrightarrow{V_I}$	**OPEN**		closed		**OPEN**	
2		**OPEN**		closed		closed	$\xrightarrow{V_O}$
3	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
4	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
5		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
6		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
Net:	$\xrightarrow{V_D}$		$\xrightarrow{V_D}$		$\xrightarrow{V_D}$		$\xrightarrow{V_D}$

Table 5.2: A single iteration of the six-step actuation cycle for a three-valve diaphragm pump.

and 1) in the five-step cycle. For this reason, the six-step cycle was originally used to operate the microfluidic processor during SNP-based DNA computations [136], although subsequent experiments using the microfluidic processor have used the three-step cycle discussed below. Like the five-step cycle, the six-step cycle is unsuitable for bubble-free pumping of fluid into initially-empty channels but is suitable for pumps with differently-sized diaphragm valves.

Attempts to further decrease the pulsatility of the three-valve pump led to the development of the three-step actuation cycle shown in Figure 5.3. In the three-step

Step		Input		Diaphragm		Output	
1	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_I}$
2		closed	$\xrightarrow{V_I}$	**OPEN**		closed	
3		closed		closed	$\xrightarrow{V_I}$	**OPEN**	
Net:	$\xrightarrow{V_I}$		$\xrightarrow{V_I}$		$\xrightarrow{V_I}$		$\xrightarrow{V_I}$

Table 5.3: Three-step actuation cycle.

cycle, a single valve is open during each step. The cycle begins in step 1 with only

the input valve open. In step 2, the input valve closes and the diaphragm valve opens simultaneously, transferring the contents of the closing input valve into the opening diaphragm valve. This transfer occurs again in step 3 as the contents of the closing diaphragm valve are transferred to the opening output valve. Finally, the output closes and the input opens as the cycle repeats and returns to step 1.

The net effect of the three-step cycle is the pumping of one *input* (not diaphragm) valve's volume forward per cycle. Since the volume of the input valve is *transferred* to the diaphragm and output valves during the three-step cycle, only an input valve's volume is available to fill the diaphragm valve in step 2 and the output valve in step 3. For this reason, three-step cycle is only suitable for pumps with equally-sized input, diaphragm, and output valves. The three-step cycle has an efficiency of $\epsilon = 1/3$ or 33%, the highest efficiency of any known cycle for a single pump. The three-step cycle also has the lowest pulsatility of any known single-pump cycle at $p = 1/3$ or 33%. This makes the three-step cycle suitable for applications that benefit from reduced pulsation. Perhaps most importantly, the three-step cycle does not demonstrate the "two steps forward, one step back" behavior characteristic of the five- and six-step cycles. When pumping into a dry channel using the three-step cycle, the fluid front moves forward every third step and never reverses. The three-step cycle is therefore very suitable for applications requiring bubble-free pumping of fluid through an initially-empty channel. Finally, the "rolling wave" nature of the three-step cycle is similar to the behavior of a common laboratory peristaltic pump,

although microfluidic peristaltic pumps can be actuated according to any of the cycles presented here [84]. Note that the analogous three-step cycle with one valve *closed* at each step, while still reminiscent of peristalsis, does not actually function as a pump. The cycle shown in Table 5.4 only moves an input valve's volume of fluid backward

Step		Input		Diaphragm		Output	
1		closed	$\xrightarrow{V_I}$	**OPEN**	$\xleftarrow{V_I}$	**OPEN**	
2		**OPEN**	$\xleftarrow{V_I}$	closed		**OPEN**	
3		**OPEN**		**OPEN**	$\xleftarrow{V_I}$	closed	
Net:	*none*		*none*		*none*		*none*

Table 5.4: Alternate (non-pumping) three-step actuation cycle.

and forward between the three pump valves. While not useful for pumping, this alternate three-step cycle could be used to mix small volumes or lyse cells held in the pump valves.

5.5 Multi-phase pumps

Interest in more efficient and less pulsatile pump actuation cycles led the the design of *multi-phase diaphragm pumps*. These structures contain multiple diaphragm pumps arranged in parallel and actuated according to a cycle that pumps the same volume of fluid forward *with each step*. Multi-phase pumps therefore have an actuation efficiency $\epsilon = 1$ and a pulsatility $p = 0$, making them well-suited for applications requiring steady flow [141].

In general, an n-phase pump containing n individual pumps is actuated using an n-step cycle. Using the three-, five-, or six-step actuation cycles characterized in the previous section, three-, five-, or six-phase pumps can be operated, respectively. Each individual pump in a multi-phase pump is "one step behind" a neighboring pump, meaning that all n pumping steps of a single-phase pumping cycle are executed simultaneously in *each step* on an n-phase pump. The volume pumped *per step* by an n-phase pump is therefore the *net* volume pumped *per cycle* by a single-phase pump using the same cycle, and the volume pumped *per cycle* by an n-pump multi-phase pump is n times the volume pumped per cycle by a single-phase pump. Remarkably, all $3n$ valves in an n-phase pump can be controlled using only *three* pneumatic lines, the same number required to control a single-phase pump. In Figure 5.4, the channel

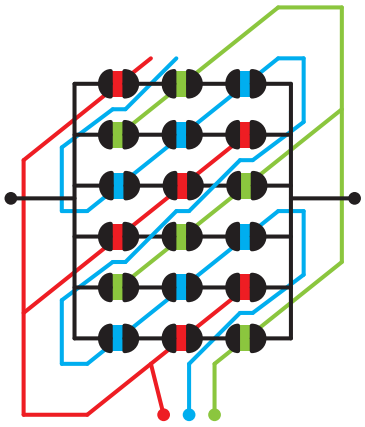


Figure 5.4: Channel layout of a multi-phase pump, showing fluid channels (black) and the three separate but coplanar pneumatic lines (red, green, and blue). Since no pneumatic lines cross, the structure can be fabricated in a three-layer (glass-PDMS-glass) device and controlled using only three pneumatic lines.

layout for a six-phase pump is presented. The three separate pneumatic lines (red,

green, and blue) connect diagonally to the 18 pump valves. By applying vacuum and pressure to the three pneumatic lines according to a six-step actuation cycle, all six separate pumps actuate in parallel with one pump performing each step of the cycle at any given time. No pneumatic lines cross each other, so all pneumatic features can be fabricated into a single wafer. The design in Figure 5.4 can be applied to multi-phase pumps containing any number of individual pumps and remain a three-layer structure.

The five-phase version of the five-step actuation cycle is shown in Table 5.5. This complex actuation cycle pumps one diaphragm valve's volume forward per step and five volumes per cycle. The simpler three-phase version of the three-step actuation cycle is shown in Table 5.6. In this cycle, one input valve's volume is pumped forward per step and three volumes are pumped per cycle. While experimental characterization of these cycles is ongoing [141], all proposed multi-phase actuation cycles have the same efficiency (1) and pulsatility (0), and the simplest cycle (the three-phase, three step) requiring the smallest number of valves (9) will probably prove to be most useful in practice.

5.6 Valve actuation in sealed sections of devices

In many applications, valves may need to operate upon fluid within a *sealed* section of a microfluidic device—a region without connections to drilled reservoirs, sippers, or other "off-device" sources of fluid. For example, the recirculating loop in the

Step	Pump		Input		Diaphragm		Output	
	1	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_O}$
	2	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
1	3	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
	4		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
	5		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
	1	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
	2	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
2	3		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
	4		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
	5	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_O}$
	1	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
	2		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
3	3		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
	4	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_O}$
	5	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
	1		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
	2		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
4	3	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_O}$
	4	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
	5	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
	1		closed		closed	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$
	2	$\xrightarrow{V_I}$	**OPEN**		closed		closed	$\xrightarrow{V_O}$
5	3	$\xrightarrow{V_D}$	**OPEN**	$\xrightarrow{V_D}$	**OPEN**		closed	
	4	$\xleftarrow{V_I}$	closed		**OPEN**		closed	
	5		closed		**OPEN**		**OPEN**	$\xleftarrow{V_O}$
Net:		$\xrightarrow{5V_D}$		$\xrightarrow{5V_D}$		$\xrightarrow{5V_D}$		$\xrightarrow{5V_D}$

Table 5.5: A five-phase version of the five-step actuation cycle from Table 5.1. The five pumps are arranged in parallel as shown in Figure 5.4.

Step		Input 1 Input 2 Input 3		Diaphragm 1 Diaphragm 2 Diaphragm 3		Output 1 Output 2 Output 3	
1	$\xrightarrow{V_I}$	**OPEN** closed closed	$\xrightarrow{V_I}$	closed **OPEN** closed	$\xrightarrow{V_I}$	closed closed **OPEN**	$\xrightarrow{V_I}$
2	$\xrightarrow{V_I}$	closed closed **OPEN**	$\xrightarrow{V_I}$	**OPEN** closed closed	$\xrightarrow{V_I}$	closed **OPEN** closed	$\xrightarrow{V_I}$
3	$\xrightarrow{V_I}$	closed **OPEN** closed	$\xrightarrow{V_I}$	closed closed **OPEN**	$\xrightarrow{V_I}$	**OPEN** closed closed	$\xrightarrow{V_I}$
Sum:	$\xrightarrow{3V_I}$		$\xrightarrow{3V_I}$		$\xrightarrow{3V_I}$		$\xrightarrow{3V_I}$

Table 5.6: Three-step, three-phase pump actuation cycle.

microfluidic processor is loaded with fluorescent input DNA from a drilled reservoir, then the reservoir valves are closed and the contents of the loop are sealed on-chip, ready to recirculate [136]. The fixed volume of fluid within a sealed section of a device limits which and how many valves can be open or closed at any given time:

Rule 3 *A valve will open within a sealed section of a device only if a neighboring valve closes.*

For example, if no valves are open within the microfluidic processor when the recirculating loop is sealed off, then no valves can be opened within the sealed processor [136]. Since the loop is sealed, no fluid is available to fill an opening valve. Attempting to open a valve in a sealed section of a device merely depressurizes the trapped fluid and generates bubbles from gases dissolved in the fluid. Similarly, if all valves are open

within the processor when the recirculating loop is sealed off, then no valves can be closed within the sealed processor.

This rule must be considered carefully when designing and operating devices that utilize valves in sealed structures like on-chip recirculating loops [135,136,141]. Every valve opened within a sealed loop must be associated with another equally-sized valve closing. This renders some of the pump actuation cycles unsuitable for use inside sealed loops. The volume pushed into the loop in Step 1 of the three-step cycle (Table 5.3) is equal to the volume pulled out of the loop in same step, and the remaining two steps do not push or pull fluid in the loop, so the three-step cycle is suitable for sealed loop pumping. In contrast, Steps 2 and 4 of the five-step cycle (Table 5.1) pull fluid out of the loop without pushing the same volume into the loop, and steps 3 and 5 push fluid into the loop without pulling the same volume out. In practice, the five-step actuation cycle *can* be used to pump inside a sealed loop, but momentarily pressurizing and depressurizing the loop in Steps 2 through 5 can lead to bubble formation and fluid leakage.

5.7 Design and control of grid processors

Monolithic membrane valves [107] were first used in complex valved grid processors for DNA-based computation. These devices were inspired by the "micro-flow bio-molecular computer" proposed by Gehani and Reif in 1999 [45] and discussed in Section 1.7. Early grid processors contained capture chambers filled with polyacry-

lamide gel sparsely decorated with Acrydite-immobilized capture oligonucleotides. These chambers were arranged in a two-dimensional grid and connected to each other by unvalved channels. By introducing fluorescent input DNA into the device and applying an electric field across the grid, the input DNA could be routed through the capture chambers, captured by sequence-specific hybridization, released by denaturation, and routed in the orthogonal direction by a different electric field. The pattern of capture oligonucleotides immobilized in the device would define the problem to be solved, and the path followed by the input DNA through the device would represent the solution to the problem. In practice, it was extremely difficult to impose a uniform and unidirectional electric field on the grid of capture chambers. Small irregularities in the device fabrication caused lateral electric currents that routed input DNA in wrong directions. The complex grid processors were also very difficult to derivatize via the standard Hjerten coating [142], so the suppression of electroosmotic flow was non-uniform across the device.

In an effort to suppress errant electrical currents in the grid processor, valves were added to the channels between the capture chambers. While the valves were ultimately ineffective at suppressing electrical current when closed, the valved grid processor remains one of the most complex devices created using the monolithic membrane valves, and its design merits discussion here. The most advanced valved grid processor is shown in Figure 5.5. The three-layer (glass-PDMS-glass) device consists of a rectilinear array of 81 capture chambers (black) interconnected by 180 valved

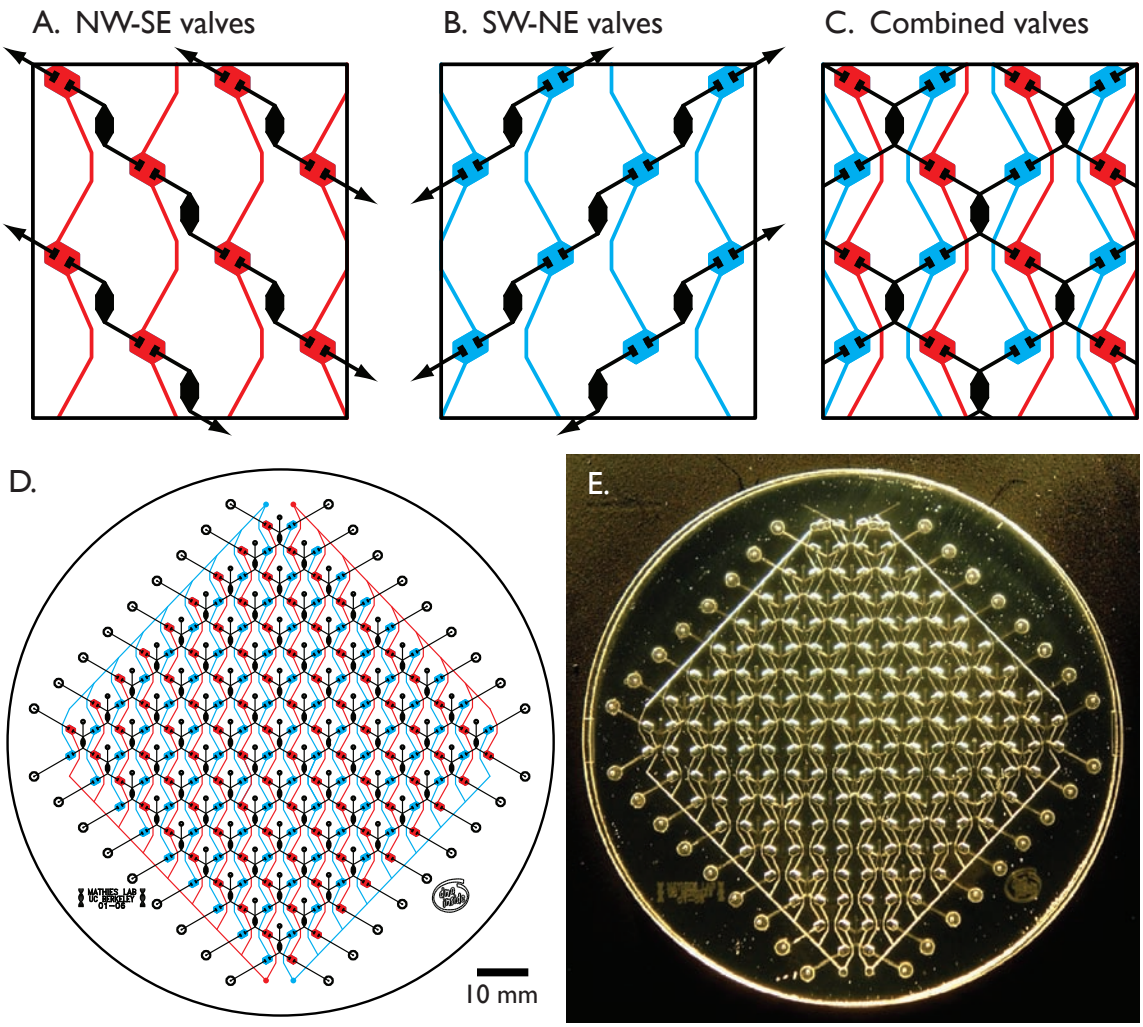


Figure 5.5: Detail diagrams showing the allowed paths of fluid flow (arrows) through the valved grid processor while red "northwest-southeast" (A) or blue "southwest-northeast" (B) valves are open. (C) The combined grid processor design. Since the two pneumatic systems (red and blue) never cross, they can be fabricated in a single pneumatic wafer. The complete grid processor (D and E) includes 180 valves actuated by only two pneumatic control lines and responsible for directing fluid flow from 36 input/output reservoirs through 81 capture chambers.

channels. Remarkably, this large number of valves requires only two pneumatic control lines because valves that actuate simultaneously are controlled together through two interdigitated networks of pneumatic control lines, each of which controls 90 valves. Figures 5.5A and B illustrates the layout of the two pneumatic control networks. The valves responsible for controlling flow in the northwest-southeast direction (red features in Figure 5.5A) are independent from the valves that control flow in the southwest-northeast direction (blue features in Figure 5.5B). When combined in Figures 5.5C, D, and E, the two networks of valves do not cross, so all 180 valves can therefore be fabricated in a single pneumatic wafer.

Although unsuitable for its intended purpose, the valved grid processor remains an interesting structure for other potential applications. For example, while the valves are ineffective at blocking electric current, the structures illustrated in Figure 5.5 could be used to route different pressure-driven flows of fluid down selected rows and columns. By including parallel banks of diaphragm pumps at the edges of the grid, a hydrodynamic implementation of Gehani's "micro-flow bio-molecular computer" could be realized. Possible computational and analytical applications for this computer are discussed in detail in the Prospects section.

5.8 Multi-layer processors

Following the technically-challenging attempts to use arrays of volumetric (gel-based) capture for DNA computations, arrayed *surface-based* capture was investigated

as a more feasible alternative. Two surface-based capture methods were investigated in parallel: a magnetic bead-based system that was subsequently used in computing and analytical applications [136], and a porous monolith-based system that used special silica-filled via holes for sequence specific capture by hybridization. While the sol gel chemistry used to form the silica monoliths was never fully optimized for DNA immobilization, the five-layer device developed for the monoliths proved the feasibility and utility of multi-layer processors.

A cross-section through a typical five-layer device is shown in Figure 5.6. The device consists of three glass wafers and two featureless PDMS membranes in an alternating "glass-PDMS-glass-PDMS-glass" stack. The top and bottom glass wafers function as pneumatic wafers, directing the flow of fluid on both sides of the middle fluidic wafer. In the monolith processor, rows of drilled holes in the fluidic wafer are filled with chemically-modified silica monoliths prior to device assembly. With the valves in the two pneumatic wafers normally closed as shown in Figure 5.6A, the monolith holes are sealed off from each other and the remainder of the device. When vacuum is applied to open the valves in Figure 5.6B, a continuous serpentine path through the row of holes is formed. Note that, in this example, the middle fluidic wafer actually has no etched channel features on either side, only drilled holes. The five-layer structure can therefore be used to control fluid flow on both sides of channel-free fluidic wafer. In practice, minimal channel features were used on the fluidic wafer in the monolith processor, as shown in Figure 5.6C. Fluid enters the

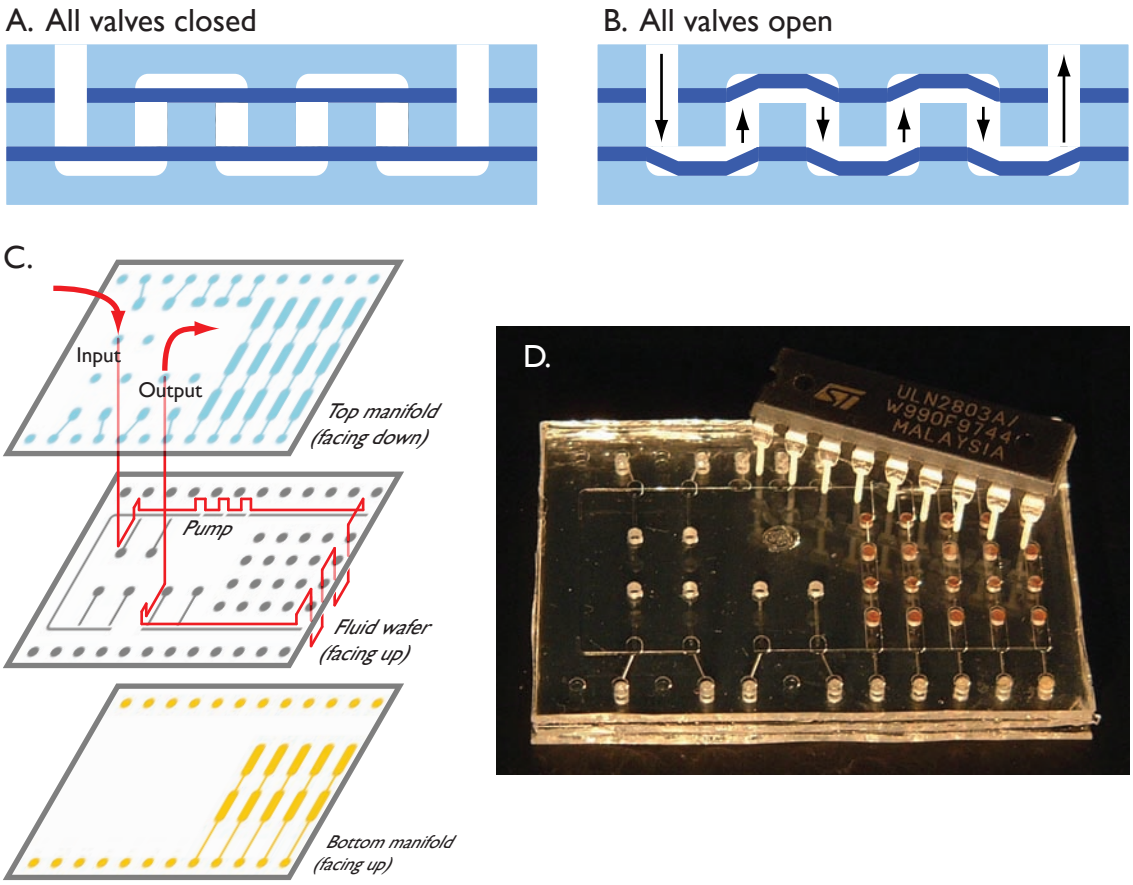


Figure 5.6: Cross-section through the five-layer processor with valves closed (A) and open (B). The device includes three etched glass wafers (light blue) and two featureless PDMS membranes (dark blue). (C) Exploded view of the five-layer processor. The red line highlights the path of fluid from a valved input reservoir, through the pump, through the selected row of four drilled holes in the fluid wafer, to the output reservoir. (D) Photograph of the five-layer processor with 3-glycidoxypropyltriethoxysilane-modified silica monoliths (orange color) in the drilled holes on the fluid layer.

five-layer structure through holes punched through the top PDMS membrane. Once on the layer of the fluidic wafer, the fluid is pumped up and down through the selected row of monoliths by valves in the top and bottom manifold wafers.

A few minor improvements would make the five-layer monolith processor a more useful structure in future studies. First, monoliths more compatible with DNA hybridization systems should be incorporated into the fluidic wafer [143]. Chemically-modified oligonucleotides could then be immobilized on selected monoliths prior to device assembly. Pre-assembly oligonucleotide immobilization is feasible because no thermal bonding is involved in five-layer device assembly. Monolith processors should also utilize recirculation, which was successfully applied to bead-based DNA capture in [136]. Perhaps the greatest potential for five-layer devices lies in their ability to incorporate unetched fluid wafers as their middle layers. Any wafer that bonds to the PDMS membrane could then be used as a middle layer, including silicon wafers containing previously-fabricated electrical sensing or actuating features.

5.9 OCW valve control software

To conclude this chapter, the computer language developed to control complex microfluidic processors is discussed. *OCW* is an interpreted computer language for controlling microfluidic valves and pumps. The name "OCW" comes from the three most important commands in the language: `o` (open a specified valve), `c` (close a specified valve), and `w` (wait for a specified amount of time). Before OCW, microfluidic

devices were controlled using an assortment of specialized, single-purpose programs written in LabVIEW (National Instruments, Austin, TX). OCW replaces these individual programs with a single program that runs OCW *scripts*, short text files that describe the order of microfluidic operations in an experiment. A new microfluidic device or assay requires only a new OCW script, and the reuse of common subroutines from earlier scripts simplifies the creation of new scripts. Since its development, OCW has been used to control microfluidic devices for SNP-based DNA computing [136], extraterrestrial amino acid analysis [110], and multiplexed latching valve characterization [140].

The OCW interpreter program was first written in Perl for computers running Linux and pre-2000/XP versions of Windows; the source code for this implementation is included in the Appendix. OCW was later ported to LabVIEW for use on Windows 2000/XP and Macintosh computers. Both versions of OCW are capable of using a computer's parallel ports to control monolithic membrane valves; the LabVIEW version can also use the digital input/output channels on National Instruments data acquisition cards for valve control.

As an introduction to the syntax of the OCW language, here is part of the script used to load beads into Chamber A of the microfluidic processor in [107]:

```
main
call close_all_valves
/ Load beads in input reservoir
stop
o4
o5
```

```
w1000
call pump_left 100
call close_all_valves
end
```

All OCW scripts must contain a main function that begins with `main` and ends with `end`. The statement `call close_all_valves` calls a subroutine named "`close_all_valves`" (defined elsewhere in the script); the subroutine, as the name implies, makes sure that all valves on the device are closed. The line that begins with a forward slash (`/`) is a comment that is printed on the screen as instructions to the operator. The program waits at the `stop` command until the operator loads the requested beads and presses ENTER. The next two lines, `o4` and `o5`, open valves 4 (the right input/output valves) and 5 (the Chamber A bus valves). The program then waits (does nothing) for 1000 ms at the `w1000` command, after which the subroutine named "`pump_left`" is called *repetitively* 100 times. Assuming that the "`pump_left`" subroutine encodes a single pump actuation cycle, repeating it 100 times using `call pump_left 100` effectively operates the pump through 100 cycles. In practice, the operator may need to escape the pumping subroutine before all 100 cycles have finished because the input reservoir may be pumped dry. In this case, pressing ENTER escapes the repeating subroutine and execution proceeds to the next line. The subroutine "`close_all_valves`" is then called once more before the program ends.

The full version of this script is included in the Appendix, along with a list of all

OCW commands and their proper syntax.

Chapter 6

Prospects

> *I can't be as confident about computer science as I can about biology. Biology easily has 500 years of exciting problems to work on, it's at that level.*
>
> —*Donald Knuth*

In the future, the intersection between computer science and molecular biology will continue to be fertile ground for innovations in both disciplines. The research presented in this thesis represents a first step toward future platforms that combine aspects of computer science and molecular biology to solve both computational and analytical problems. In this final section, five aspects of this proposed *programmable microfluidic processor* are considered. The processor should be *generic*, a single device that is useful for both computations (like the single-nucleotide polymorphism-based DNA computation presented in Section 4) *and* assays (like the haplotype determination assay described in this section). The processor should support *evolution-*

ary molecular computation as a powerful alternative to the "all possible answers" paradigm of DNA computing. It should function as a *nano-scale foundry* for the fabrication of self-assembled nanostructures. Like the pneumatic player piano, it could be programmed and controlled by on-chip *pneumatic logic* instead of off-chip computers. Finally, the processor could be modeled after traditional *cellular automata* and be structurally simple yet versatile and powerful. A programmable molecular processor that combines these five aspects could be a powerful tool for solving a vast array of computational and analytical problems.

6.1 Haplotyping on the molecular processor

The hope that DNA computing techniques would also find uses in other fields is almost as old as DNA computing itself. Two years after he demonstrated the first DNA computation, Len Adleman wrote, "it is the author's hope that the ideas used in designing and programming molecular computers will be of direct (non-computational) value in biology, chemistry and medicine" [23]. Three years later, Laura Landweber described a "killer application" for DNA computing:

> ... an application that fits the DNA model, cannot be solved by the current or even future electronic machines, and is *important*... The key new idea is to use DNA computation to operate on *unknown* pieces of DNA [144].

In this vision, the unknown DNA could be amplified genomic DNA, and the "computation" could be the screening the individual for genetic disease markers, fingerprint-

ing the individual for forensic applications, or sequencing a portion of the individual's genome.

Progress toward realizing Landweber's vision of a analytical "killer app" for DNA computing has been slowed by the use of multi-base bits in nearly all DNA computations. Since Adleman's seminal computation [11], most DNA computations have been performed using constant sequences of 15–20 bases to represent single binary bits. Used to increase the stringency of hybridization, these artificial multi-base bits have no analogue in living organisms in which *single base* variations are often important. Landweber recognized this problem and proposed that genomic DNA could be translated into a multi-base-bit format, with each base in the genomic DNA replaced by 15- or 20-base sequences in the computational DNA [144]. This is unfeasible because no existing genetic mechanism can perform this translation, and only short lengths of genomic DNA would be amenable to the 15× or 20× increase in length that would accompany translation. As a result of this reliance on multi-base bits, no techniques developed for DNA computing have yet been successfully applied to analytical (non-computational) tasks.

Since it can recognize single-base mismatches, the molecular processor presented in Section 3 can operate directly on amplified genomic DNA. This is a major step toward realizing Landweber's "killer app," as *any* DNA computation that could be performed using single-base bits on the molecular processor could also find use as a genetic assay. Indeed, by merely substituting amplified genomic DNA for the synthetic DNA used

in the computation described in Chapter 3, a genotyping or haplotyping assay can be performed. While experimental validation of haplotyping on the molecular processor is still in progress [145], an outline of the procedure is presented here to support the claim that *general purpose* (computational and analytical) molecular processors are both feasible and extremely useful.

The process of haplotyping using the molecular processor is diagrammed in Figure 6.1. The molecular processor is programmed with magnetic capture beads derivatized with 15- to 20-base oligonucleotides complementary to the regions of genomic DNA containing the polymorphic bases of interest. In step 1 of Figure 6.1, the capture beads for biallelic SNP 1 are complementary to genomic DNA containing either C (chamber N) or T (chamber O) at the polymorphic base, and chamber pairs B and C, E and F, and K and L contain capture beads complementary to the remaining three SNPs. The region of interest in a genomic DNA sample is PCR-amplified using a fluorescein-labeled forward primer and biotinylated reverse primer. The resulting double-stranded amplicon is conjugated to magnetic streptavidin beads, and the fluorescein-labeled strand is eluted using 25% ammonium hydroxide solution [146]. The resulting single-stranded DNA is desiccated in a vacuum centrifuge and resuspended in hybridization buffer [147]. This solution becomes the "input" DNA and is pumped into chamber A of the molecular processor in Step 2.

In step 3 of Figure 6.1, single-stranded fluorescent amplicon is recirculated through the SNP 1 capture beads in chambers N and O. Fluorescence imaging is then used

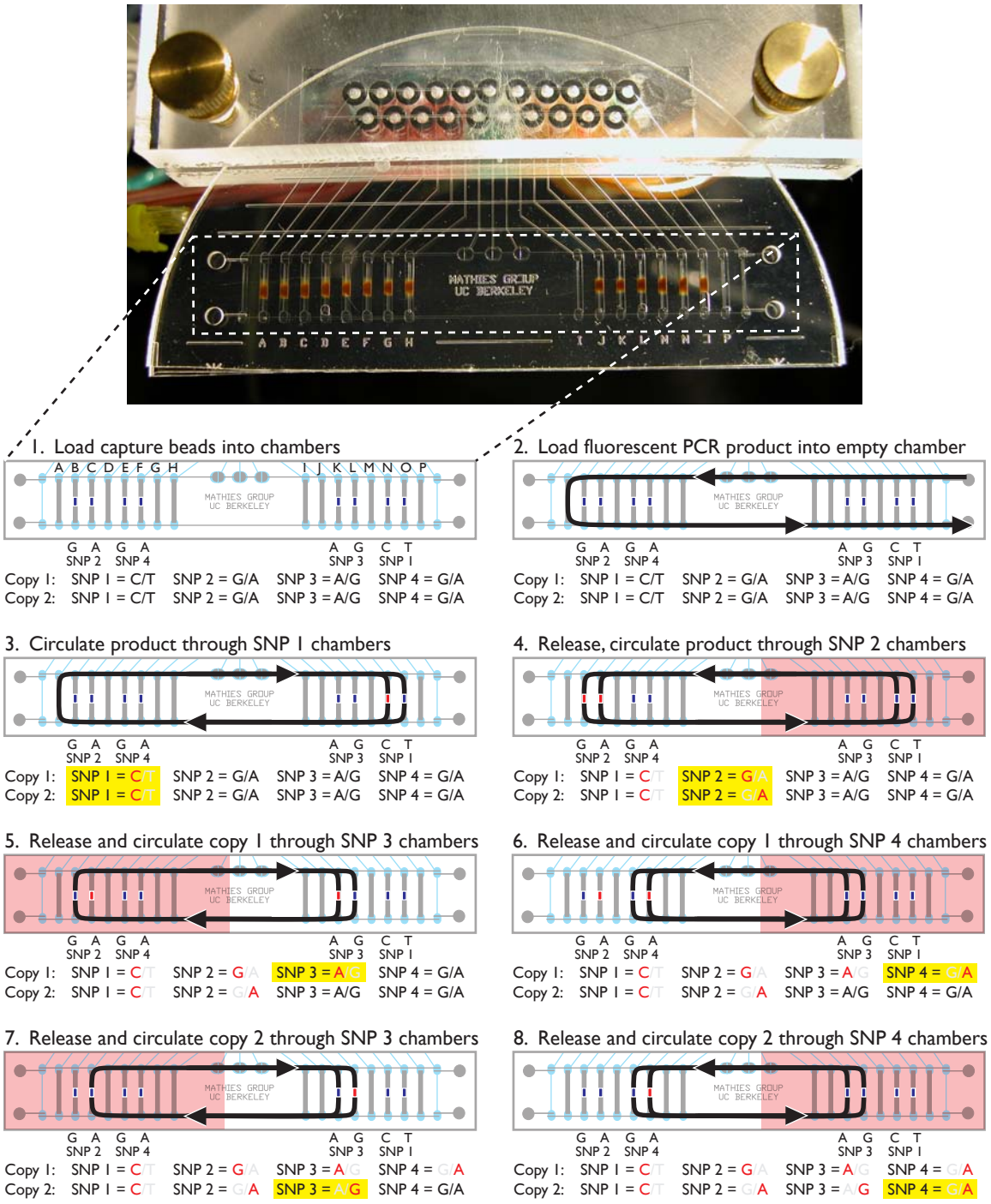


Figure 6.1: Haplotyping on the molecular processor.

to determine which of the two bead boluses captured the amplicon. In this example, the measured fluorescence is higher on the beads complementary to C at SNP 1, indicating that both copies of the individual's genome must have C at SNP 1 and the individual is *homozygous* for that SNP. In step 4, the amplicon is released by heating the right-hand side of the device and recirculated through the SNP 2 capture beads on the left-hand side of the device. In this example, the measured fluorescence is *equal* in both bead boluses, indicating that the individual is *heterozygous* for SNP 2. At this point, amplicons from the two copies of the individual's genome are spatially separated on the two SNP 2 bead boluses, and each copy can now be operated upon separately to obtain the individual's haplotype. In step 5, only the copy of the genome containing G at SNP 2 is released thermally from the SNP 2 beads and recirculated through the two SNP 3 capture chambers. Since only one copy of the genome is being operated upon, this and subsequent capture steps must result in either one or the other bead bolus being more fluorescent, never both being equally fluorescent. The observed pattern of fluorescence indicates that this copy of the genome has A at SNP 3 (step 5) and A at SNP 4 (step 6). Finally, the other copy of the genome (the copy containing A at SNP 2) is released from the SNP 2 beads and recirculated through the SNP 3 beads in step 7 (G at SNP 3) and the SNP 4 beads in step 8 (A at SNP 4). The path followed by the amplicons of the two copies of the genome can then be used to reconstruct the haplotype of the individual: heterozygous at SNPs 2 and 3, with SNP 2 = G and SNP 3 = A on copy 1 and SNP 2 = A and SNP 3 = G

on copy 2, and homozygous at SNPs 1 and 4.

The molecular processor may offer a number of benefits over conventional tools for haplotyping like microarrays. Since each SNP is analyzed separately, regions containing SNPs flanked by GC-rich regions can be captured at a higher temperature than regions containing SNPs flanked by GC-poor regions. In a more-parallel microarray assay, differences in duplex stability across the various SNPs can complicate readout and are usually minimized by designing capture oligonucleotides with a narrow range of melting temperatures (possibly at the expense of decreasing the destabilization caused by the SNP mismatch). In the molecular processor assay, capture oligonucleotides can be designed solely to maximize the difference between the matched and mismatched SNPs, regardless of the overall melting temperatures of the different SNPs. While the serial operation of the microfluidic processor may make it slower than microarray-based tools, interrogating the SNPs in the order of increasing likelihood of heterozygosity would minimize the average number of steps required to haplotype a sample. Alternately, the molecular processor could be used as a "front end" for other analysis methods: the processor could detect the first heterozygous SNP and separate the amplicon copies based on that SNP, then send these copies to downstream to thermal cycling and separation steps for conventional genotyping. These two genotyping results could then be combined to reconstruct the haplotype of the individual. Finally, since no "incorrect answers" are being eliminated, few if any post-capture rinsing steps should be required while haplotyping on the molecu-

lar processor. Since the step-to-step transfer efficiency of the processor is inversely proportional to the amount of bead rinsing required [136], reducing or eliminating rinsing should increase the efficiency of the processor, decrease the amount of time required for capture, and make large-scale haplotyping analysis feasible on the molecular processor.

When integrated into the molecular processor, previously-unrealized proposals for DNA-based computing may also find new life as components of genetic assays. For example, consider Roweis' proposed "compound separator" discussed previously in Section 1.7.2. Resembling fractionation columns used in oil refineries, the compound separator consists of several individual capture/release components connected together in parallel. Perfectly-complementary DNA must be successfully captured and released several times before emerging from the "positive" end of the compound separator, and mismatched DNA must *avoid* capture several times before emerging from the "negative" end of the separator. Though intended for computational applications using multi-base bits, the refinery-like compound separator could be extremely well-suited for sorting out DNA based on single-nucleotide polymorphisms. And while the microfluidic technologies necessary to build it were not available in 1996, a microfluidic compound separator could easily be fabricated using the valve and pump structures presented in this thesis. Microfluidic compound separators could improve single-base mismatch-specificity in a variety of hybridization-based computational and analytical applications, including the haplotyping assay presented here. Note

that the compound separator is only one of many proposals for DNA computing that were technologically infeasible when proposed but could now be integrated into the microfluidic processor platform and applied to a limitless number of computational and analytical problems.

6.2 Evolutionary computing

Like the use of multi-base bits, the "all possible answers" approach to DNA computing also has no direct analogue in living organisms. Evolution does not select for the fittest of a species by first generating "all possible mutations." The huge size of organisms' genomes precludes such an exhaustive, brute-force approach to finding fitness. Instead, natural selection follows an iterative process in which occasional mutations sample the huge landscape of "all possible mutations," advantageous mutations are passed on to future generations, and deleterious mutations die out. While the resulting organisms can never claim to be the fittest of "all possible organisms," the complexity and diversity of life attests to the success of evolution.

The same problems that make the "all possible answers" approach unsuitable for living organisms make it ultimately unsuitable for DNA computing as well. There may be only about sixty 15-base DNA sequences that are sufficiently different to serve as bit sequences in an "all possible answers" DNA computation [29]. Sixty sequences is enough for a 30-bit computation, a computation only 10 bits larger than the current largest DNA computation and solvable in seconds using optimal algorithms on con-

ventional computers [30]. While computations using single DNA bases to represent binary bits were presented in this thesis, the difficulty of distinguishing single-base mismatches in long DNA strands limits the size of problem solvable using single-base bits in an "all possible answers" computation to about 8 bits [136]. Clearly a new computing method is necessary if DNA computers are to have any hope of competing with their silicon counterparts.

As discussed in Section 1.3, in 1948 John von Neumann suggested that the molecular machinery of evolution had far more computing potential than the most advanced electronic computers. He described a hypothetical reproducing computer containing DNA-like instructions that were modified at random, "lethally as a rule, but with a possibility of continuing reproduction with a modification of traits" [8]. This idea catalyzed the development of *in silico* evolutionary algorithms in computer science. Then in 1989 and 1990, three labs independently used *in vitro* molecular evolution to create novel ribozymes with specific catalytic behavior [148–150]. In 1996, Laura Landweber proposed that these *in vitro* evolution methods could be used to solve computational problems [151]. Recent results prove that these molecular evolution techniques can also be miniaturized and automated using the valve and pump structures presented in this thesis [135]. In summary, all of the techniques and tools required to realize von Neumann's vision and construct a *evolutionary molecular computer* now exist.

How would an evolutionary molecular computer improve upon existing "all possible answers" DNA computers? Figure 6.2 shows a conceptual comparison between

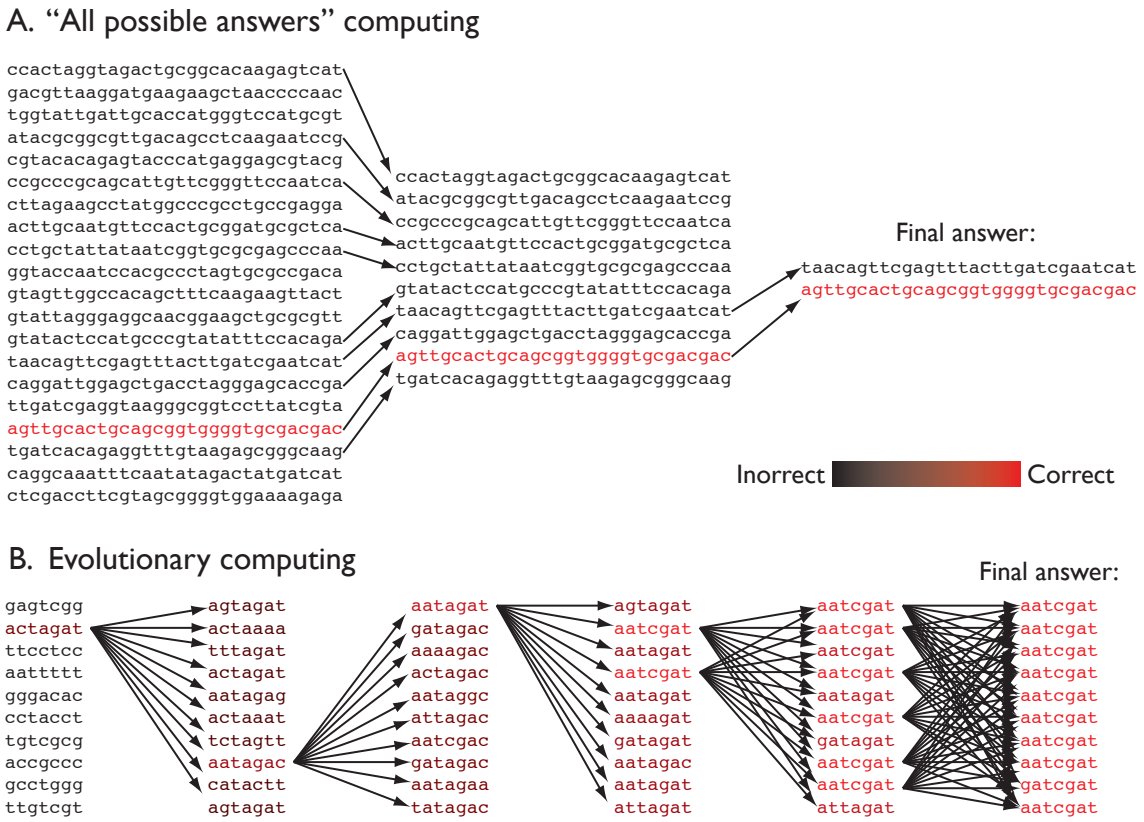


Figure 6.2: Comparison between "all possible answers" (A) and evolutionary (B) DNA computing. Sequence color represents the "correctness" of the strand, with black sequences being totally incorrect and red sequences being totally correct.

the "all possible answers" (A) and evolutionary (B) paradigms of DNA computing. In the first step of the "all possible answers" computation, the sequence representing the single correct answer (red) is vastly outnumbered by other sequences representing incorrect answers (black). Incorrect strands are discarded during a series of capture-and-release steps, and the "final answer" consists of the surviving correct strands along with any incorrect strands that may be present because of false positives. In the evolutionary computation shown in Figure 6.2B, no attempt is made to create "all possible answers" in the first step. Instead, a small number of randomly-generated oligonucleotides seed the evolutionary process. In contrast to the starting DNA in the "all possible answers" approach, *none* of these seed oligonucleotides are likely to represent purely-correct answers. Also, since the evolutionary computation is less digital than the "all possible answers" computation and does not rely on constant multi-base sequences to represent binary bits, the oligonucleotides used in an evolutionary computation can be much shorter than the input oligonucleotides used in (A). The fittest or most-correct seed oligonucleotides are selected, amplified with occasional mutations, and tested again for fitness or correctness. With each generation in Figure 6.2B, the average "correctness" of the oligonucleotides increases. By decreasing the rate of mutation in the final few steps, the few fittest or most-correct oligonucleotides can be amplified or enriched into a large, easily-detectible quantity. Note that even though slightly-less-optimal sequences may still be present in the final answer, they still represent fairly optimal solutions and are vastly outnumbered

by the more-optimal sequences. In summary, *an "all possible answers" computation is like finding a correct needle in a haystack of wrong answers, and an evolutionary computation is like breeding the answers until all of the answers are correct.*

The greatest single barrier to realizing this vision of evolutionary computation concerns the selection of the fittest or most-correct DNA in each generation. It is much easier to select strands with, for example, catalytic behavior than strands that represent correct answers to binary logic problems the traveling salesman or Boolean satisfiability. Despite this difficulty, computing systems based on the enzymatic manipulation of RNA have already been used to solve chess problems [152, 153]. Ribozymes evolved to operate upon other RNA strands in a sequence-specific manner could demonstrate feedback and be used to solve binary (base-specific) logic problems in an evolutionary computer. Even if the microfluidic evolutionary computer could not solve binary logic problems, it could still prove useful for solving non-digital optimization problems like finding optimal sequences for gene expression microarrays [154]. Regardless of their prospects for computing, microfluidic processors for *in vitro* evolution could be well-suited for creating new enzymes, developing novel therapeutics, and answering questions about the earliest forms of life on earth.

6.3 The molecular processor as a microfluidic nano-foundry

Another promising field of research in DNA computing involves self-assembling nanostructures, as discussed in Section 1.6.3. These DNA nanostructures show great promise in both computational and materials science applications: nanostructures can self-assemble according to rules that encode a problem to be solved [36], and macro-scale materials made "from the ground up" using nanoscale components could have interesting and useful properties [155].

Existing methods for DNA nanostructure self-assembly suffer from defects caused by assembly errors. For example, in tile-based DNA nanostructures, partially-mismatched hybridization can lead to the incorporation of an incorrect tile into the growing nanostructure [41]. Like a defect in a crystal, the erroneous tile can cause subsequent tiles to be incorporated incorrectly, and the resulting nanostructure will contain a large region filled with incorrect tiles. Figure 6.3A shows an example of a typical error in a tile-based DNA nanostructure. The accidental incorporation of a single "red" tile in a row of "blue" tiles causes many additional tiles to be added incorrectly. These errors are an inevitable consequence of the "one pot" nature of nanostructure self-assembly: even though the DNA sequences are designed carefully to minimize unintentional incorporation of erroneous strands, all strands are present in the self-assembling reaction at the same time, and for nanostructures with more

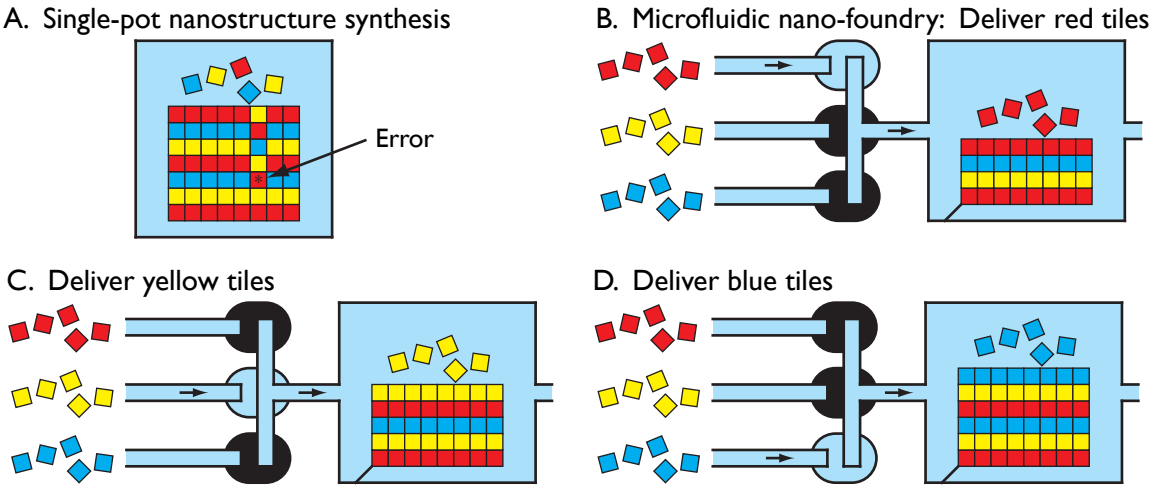


Figure 6.3: (A) Conventional "one pot" assembly of a DNA nanostructure with three alternating kinds of tiles (red, yellow, and blue). Colored squares represent DNA tiles (see section 1.6.3) and the asterisk (*) indicates the erroneous addition of a red tile in a blue row. (B-D) Operation of a microfluidic nano-foundry for DNA nanostructure assembly. Bus valves deliver each kind of DNA tile to the growing nanostructure separately without errors, creating a red row of tiles (B), then a yellow row (C), and finally a blue row (D) before the cycle repeats itself. Based on a proposal in [156].

than two types of tiles, the number of incorrect tiles available for incorporation into the nanostructure always exceeds the number of correct tiles.

In 2005, Koutaro Somei proposed that microfluidic devices could be used to reduce or eliminate assembly errors associated with DNA nanostructure assembly [156]. An implementation of his proposal using monolithic membrane valves and pumps is diagrammed in Figure 6.3B, C, and D. Instead of being combined in "one pot," different DNA tiles are kept separate in on-chip reservoirs. These tile reservoirs are connected via a valved manifold to a chamber containing immobilized seed tiles. In Figure 6.3B, red tiles are delivered to the growing nanostructure. Since no erroneous tiles are present, no substitution errors can occur. In steps C and D, yellow and blue tiles, respectively, are delivered separately to the nanostructure before the assembly cycle repeats itself. Along with reducing or eliminating errors, the microfluidic nano-foundry shown in Figure 6.3 could also vastly simplify the task of designing the DNA sequences used in constructing the nanostructures. Since the *order of delivery* of the tiles (not just their DNA sequences) determines the overall morphology of the nanostructure, a small number of different overlapping sequences could be shared among many different kinds of tiles. Like "Lego" building blocks, these tiles could be assembled into an infinite variety of different nanostructures, with the particular nanostructure formed determined by the order of delivery of the tiles to the growing nanostructure.

This microfluidic DNA nano-foundry could be useful for more than just perform-

ing computations and assembling nanostructures. Recently, an alternative method of nanostructure assembly was demonstrated that used 7000-base viral genomic DNA as a scaffold [43] This single-stranded scaffold wound throughout the nanostructure; it was held in the desired shape using short "staple" oligonucleotides, each of which kept two regions of viral DNA in close proximity. By designing and synthesizing several of these "staple" oligonucleotides, the viral DNA can be arranged and held in a huge variety of shapes: 100-nanometer squares, rectangles, triangles, stars, even world maps and "happy faces" [43]. The shape assumed by the viral DNA is a function of the sequences of both the viral DNA and the added "staple" oligonucleotides. It is possible that DNA from an unknown virus could be combined with known "staple" oligonucleotides to generate a unique folded shape. The shape could then be compared to the folded shapes of known viruses, and the unknown virus could be identified. Small sections of larger genomes (bacterial, human, etc.) could similarly be folded and analyzed. The microfluidic DNA nano-foundry would be an excellent platform for this research. Immobilized genomic DNA (or immobilized amplicons) could be subjected to different "staple" oligonucleotides one-kind-at-a-time using the microfluidic structure in Figure 6.3 while monitoring for the formation of new folded structures within the DNA. Folded shapes or motifs formed in the DNA after the addition of each unique "staple" oligonucleotide could then be correlated with specific sequences within the genomic DNA.

Perhaps the most significant barrier to performing this nanostructure-based se-

quence analysis is the current reliance on atomic-force microscopy for determining the morphology of DNA nanostructures. As mentioned in Section 1.6.3, alternatives to AFM have been proposed [36]. In 2005, Rebecca Schulman proposed that DNA nanostructures could be designed that stay disassembled until a *single nucleating seed tile* is introduced [157]. After introducing the seed tile, the nanostructure self-assembles spontaneously into a micron-size conglomerate that could be detectible by standard optical microscopy. It seems reasonable that "stapled" genomic DNA could also self-assemble into a visible conglomerate, and the appearance of conglomerates would indicate a sequence homology between the genomic DNA and the "staple" oligonucleotides. Alternately, dye- or nanocrystal-labeled "staples" could be used for fluorescent detection of the DNA nanostructures. Energy transfer dye pairs could provide detailed information about the homology of the nanostructure: detection of emitted light of a certain wavelength would be indicative that the donor and acceptor "staples" are in close proximity. These and other non-AFM methods for determining the presence and homology of DNA nanostructures should help make nanostructure self-assembly a useful tool in a variety of computational, materials science, and analytical applications.

6.4 Pneumatic logic in the molecular processor

A truly programmable molecular processor would include the *logic* of processor operation on-chip in a programmable format. The multiplexed latching valves pre-

sented in Section 4 are a major step toward this goal. By using pneumatic logical structures to control device operation on-chip, the number of off-chip controllers required to operate a device can be significantly reduced. But multiplexed latching valves still require *some* off-chip controllers, typically solenoid valves actuated by a computer. These off-chip controllers typically occupy much more space and consume much more power than the microfluidic devices they control. As microfabrication methods become more cost-effective, these off-chip controllers may soon cost more than the microfluidic devices they control. The size and power consumption of these controllers limits the portability and field usability of microfluidic devices, and the cost of these controllers limits their use in point-of-care applications.

It could be possible to operate complex microfluidic devices using *no* off-chip controllers—no solenoid valves or computers. In such a device, *all* microfluidic operations (opening and closing valves, pumping, routing fluids, etc.) would be programmed into and controlled by on-chip pneumatic logic circuits. Like microprocessors powered by constant "TTL high" (5 V) and "TTL low" (0 V) sources, these devices would be powered by constant vacuum and pressure sources. Device operation would be initiated by a signal, perhaps the presence of liquid in a reservoir or a finger blocking a drilled hole. The on-chip valves would then operate themselves according to the program required for the given assay before returning to a "ready" state.

Pneumatic valves arranged in "feedback circuits" could play an important role in

this fully-autonomous microfluidic processor. In the example shown in Figure 6.4, two valves are configured so that each one controls the other. Applying vacuum

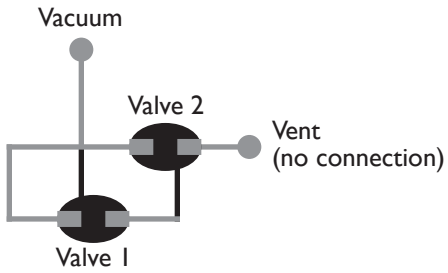


Figure 6.4: "Feedback circuit" for generating periodic valve actuation from a constant vacuum.

to the circuit opens valve 1, which depressurizes the control for valve 2 and opens valve 2. However, opening valve 2 vents most of the vacuum holding valve 1 open, so valve 1 closes. This interrupts the vacuum to the control of valve 2 and closes valve 2. With valve 2 sealed, the vacuum can again depressurize the control for valve 1 and open valve 1, and the cycle repeats itself. This is one of many possible feedback circuits that could be used to open and close a working valve in a periodic manner. For example, three feedback circuits that are 60° out of phase with each other could be used to actuate three valves according to a three-step pumping cycle. In this manner, constant vacuum and pressure sources could power oscillatory on-chip feedback circuits that in turn actuate working valves in the pattern required for a given assay.

The operation of some devices could be so complex that no arrangement of these feedback circuits would be adequate. In this case, the pneumatic player pianos dis-

cussed in Section 1.2 could offer an alternative method for programming pneumatic logic. Player pianos use punched paper programs to store songs. The paper normally seals an intake hole, which allows a vacuum to develop in a small volume inside the piano. A punched hole opens an intake hole and vents a small vacuum inside the piano, which in turn actuates a valve that depressurizes a membrane and plays a note on the piano [4]. This mode of actuation—using a vacuum to sense a blockage at an intake hole—can be used to actuate microfluidic valves. Figure 6.5 shows a pneumatic circuit that closes a valve when a drilled intake hole is open and opens the valve when the intake is blocked. In Figure 6.5A, vacuum applied to the circuit is vented through the intake, and the working valve stays closed. When the intake is blocked in Figure 6.5B, vacuum is transferred to the control of the working valve and the working valve opens. Figures 6.5C and D show frames from a video of a player piano-style valve during operation. In the top photos, the intake hole is open and the piano-style valve is closed. When a clear piece of plastic is used to block the intake hole, the piano-style valve opens. Finally, Figure 6.5E shows that a fingertip can also be used to block the intake hole and open the valve.

By replacing the piece of plastic in Figure 6.5 with a punched sheet of paper and feeding the punched holes across the intake hole, the player piano-style valve could be actuated according to any program punched in the paper. Any number of on-chip piano-style valves could be controlled in this manner. Even the most complex microfluidic operations could be encoded in the pattern of punched holes. *It*

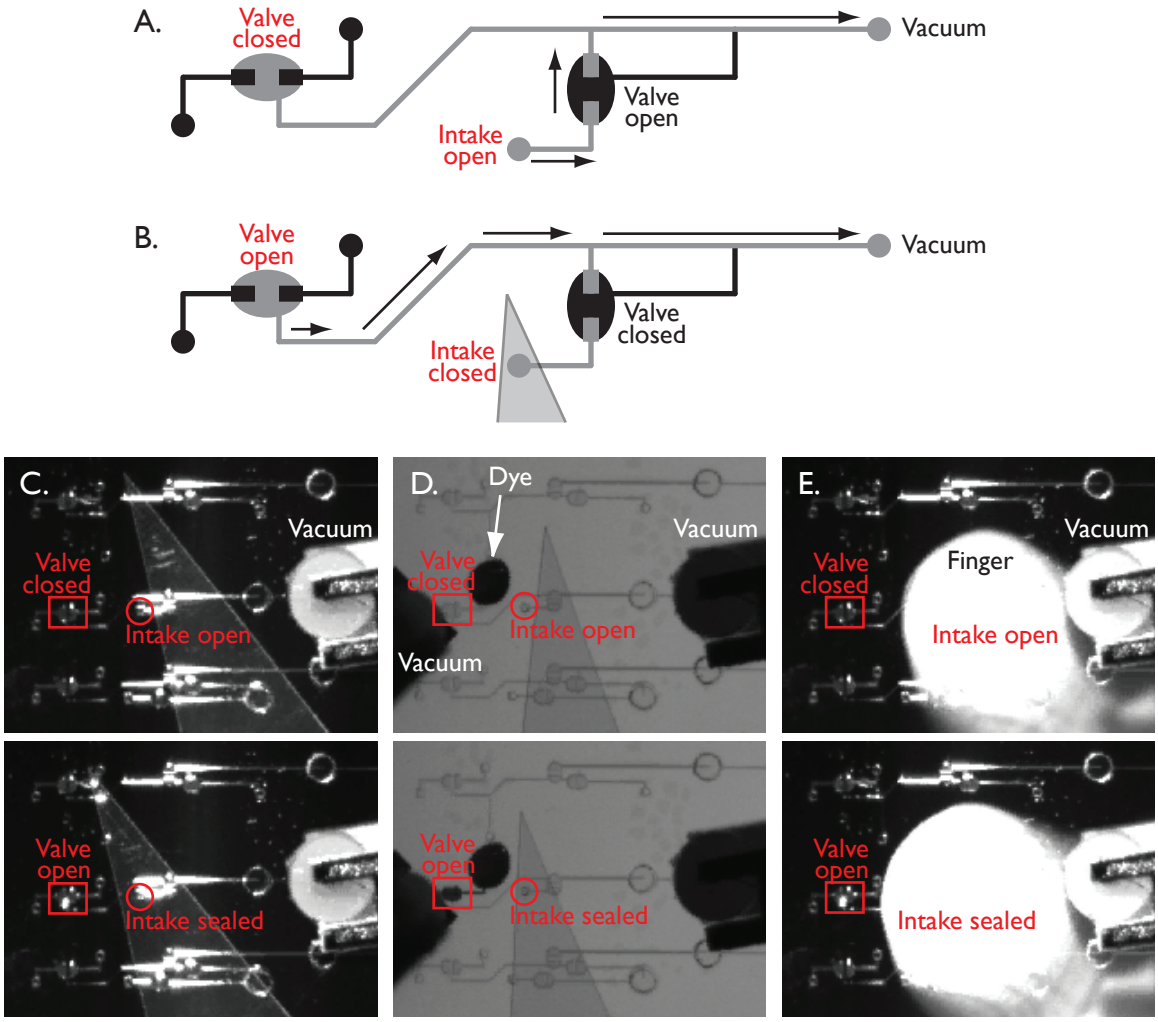


Figure 6.5: Design of a closed (A) and open (B) player piano-style valve during actuation. Frames from a video showing a player piano-style valve being actuated by a clear piece of plastic (C and D) and a fingertip (E).

is important to note that this inexpensive punched paper mechanism would replace all *off-chip solenoids and computers required to actuate a device—the device would become its own fully-programmable computer.* There are undoubtedly many applications in which off-chip computers are *required* for operations like data acquisition, and moving device control in these applications from the off-chip computer to an on-chip pneumatic computer would reap few benefits. But there are also applications in which even the crudest computer or solenoid valve manifold would be too expensive, yet the desired assay requires complex control of on-chip valves and pumps. In these applications, this punched paper controller could be an extremely cost-effective way to program and control on-chip fluidic operations.

6.5 Microfluidic cellular automata

The earliest electronic computers were special-purpose machines—they were designed to perform one specific computation, and designing a computer for a new application usually required building a whole new computer from the ground up [3]. Computers became ubiquitous only after *generic, programmable* computers were developed. The same computer can now be used to run a huge variety of different programs, and a new application now require only a new program. Microfluidic analysis devices are still at the same special-purpose stage where computers were five decades ago. There is no generic, programmable lab-on-a-chip, and efforts to develop new microfluidic assays are unnecessarily slowed by the need to both build a suitable

device for the assay and determine how to run the assay on the device.

Programmable processors for DNA computing were proposed in 1999 by Gehani [45]. As discussed in Section 1.7, Gehani's proposed device would have required hundreds of independently-controlled valves and pumps—a level of microfluidic large-scale integration that was not experimentally feasible in 1999. Microfluidic processors have since been demonstrated that use electrophoresis to route charged species like DNA [158] and electrowetting to route droplets in an immiscible phase [159], but no generic microfluidic processor suitable for use with any common analyte or fluid system has been demonstrated. Today, Gehani's vision of a programmable DNA processor could be realized using the microfluidic fluid control structures presented in this dissertation. The resulting *programmable microfluidic processor* would be useful for both computational and analytical applications. This generic platform could perform any number of different assays simply by providing different inputs and running different programs *on the same generic microfluidic structure.*

Following Gehani's original proposal [45], the processor described here contain a rectilinear array of chambers interconnected by channels. Gehani suggested that fluid flow through the chambers could be controlled using valves in the channels connecting the chambers; this approach was used in the grid processor described in Section 5.7. However, the 180 valves in the grid processor share only two common pneumatic lines and cannot be independently actuated. Since the programmable microfluidic processor must have independently-controllable valves (since fluid flow along any

conceivable path between any two chambers is possible), the grid processor is not suitable as a programmable molecular processor, but a version with independently-controllable valves *is* suitable. The four-chamber microfluidic processor design shown in Figure 6.6A has twelve independent valves that must be controlled independently.

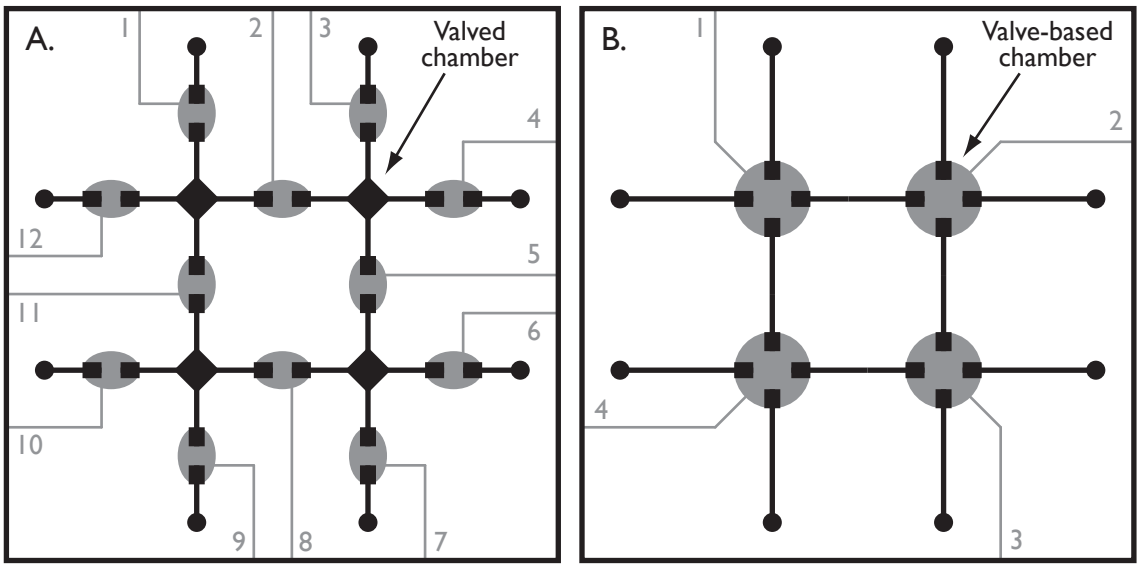


Figure 6.6: Comparison of grid processor designs using valved-chambers (A; discussed in Section 5.7) and valve-based chambers (B). Using four-way valves as chambers decreases the number of independent valves required to control a four-chamber processor from twelve in A to only four in B.

In general, an n-cell microfluidic cellular automaton based on the design in Figure 6.6A requires $2n + 2\sqrt{n}$ independent valves to operate.

An alternative design for a programmable microfluidic processor is shown in Figure 6.6B. In this design, the valves *are* the chambers: four-way valves are interconnected by valveless fluid channels. By opening or closing the valves in the desired pattern, fluids can be routed in any conceivable pattern on-chip. An n-element programmable

microfluidic processor based on the design in Figure 6.6B requires only n independent valves to operate. By using multiplexed latching valves [140], a 1000-element processor based on the design in Figure 6.6B could conceivably be operated using only 11 pneumatic control lines. While both designs in Figure 6.6 may find use in the fabrication of microfluidic cellular automatons, the simpler valve-based chamber design in Figure 6.6B seems to be superior and is used in the following discussion of sample programs for programmable microfluidic processors.

The use of valves as chambers in a programmable microfluidic processor may have an unexpected benefit: *the processor can be programmed and controlled like a cellular automaton.* As presented in Section 1.4, cellular automata are grids of hypothetical cells, each of which can have one of two different states, TRUE or FALSE (or, visually, black or white). The relevant aspect of cellular automata is that *complex patterns in the cells can be created and manipulated using very simple rules.* Simple programs like "color a cell black if two of its neighboring cells are black" can lead to very complex behavior in the overall automaton. Translated to the programmable microfluidic processor, programs like "open a valve if two neighboring valves are open" could perform useful operations like mixing or serial dilution. These programs are simple enough to be controlled by pneumatic logic on-chip, but the resulting behavior is complex enough to be of practical use in many possible circumstances. To highlight this potential, the novel valve-based programmable microfluidic processor shown in Figure 6.6B is referred to here as a *microfluidic cellular automaton* and the valves are

called *cells*.

In the rest of this section, a few programs for operating a microfluidic cellular automaton are presented and discussed. These programs serve only as examples of how basic microfluidic operations (mixing, serial dilution, etc.) can be performed in the microfluidic cellular automaton. It is important to note that the *design* of the microfluidic cellular automaton remains unchanged in each of these examples; only the *program* of valve actuation changes from one application to another.

A program for mixing the fluid contents of two cells in a microfluidic cellular automaton is presented in Figure 6.7. In Step 1, two open cells (four-way valves) are filled from the east and the west with the fluids to be mixed (blue and red); all remaining cells in the region are closed (light gray). By closing the originally-open cells and opening two neighboring valves simultaneously, the blue and red fluids are routed into neighboring cells in Step 2. When the cell containing blue fluid closes and a cell south of the red fluid opens in Step 3, the red fluid is routed south and the blue fluid is routed east into the cell previously occupied by the red fluid. Since fluid flow is laminar within the microfluidic cellular automaton and theoretically no flow occurs at the channel or cell wall, a thin shell of red fluid remains in the cell filled by blue fluid in Step 3. As this clockwise pumping of fluid continues during Steps 4 through 6, these shells of red and blue fluid grow so thin and intertwined that diffusion between the shells mixes the two fluids by Step 7. Note that Steps 4 through 7 can be repeated as many times as necessary to ensure complete diffusional

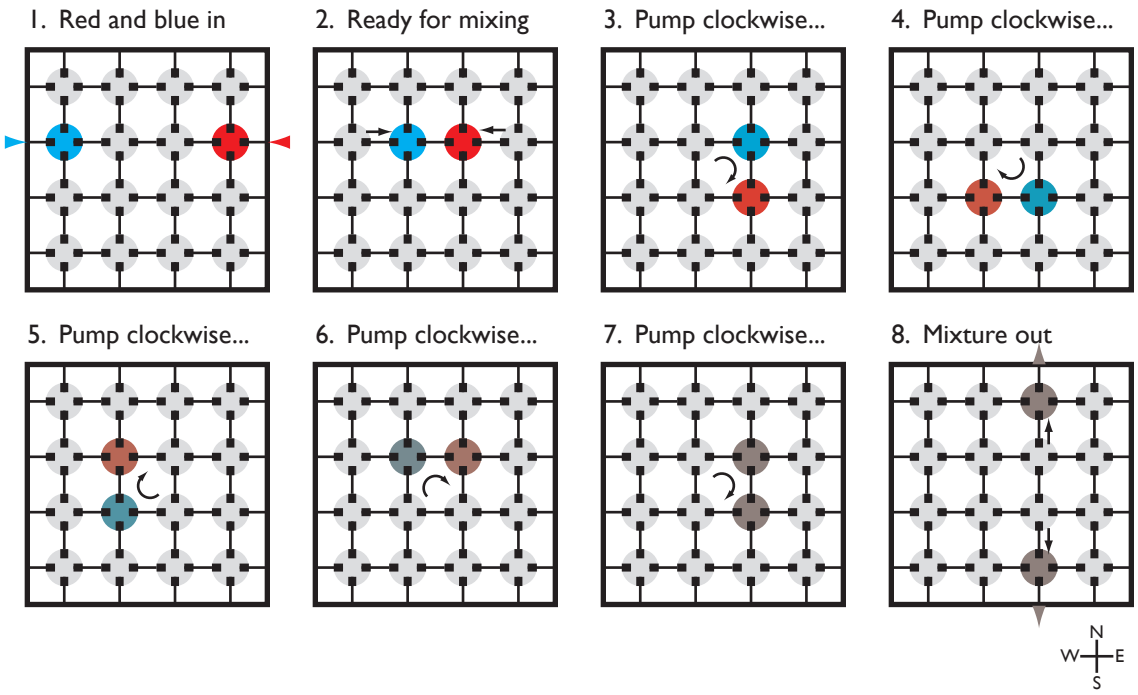


Figure 6.7: Sample program for mixing the contents of two cells in a microfluidic cellular automaton. Closed cells (four-way valves) are colored light gray; open cells are colored according to their fluid contents (red or blue fluid, or a mixture of the two). Arrows indicate the direction of fluid flow as cells are opened and closed.

mixing of the two fluids. Finally, in Step 8 the two cells containing the mixed fluid are closed as two other cells are opened, routing the mixture north and south to other cells for further processing or analysis.

The mixing program shown in Figure 6.7 requires only four cells to mix the contents of two cells; the twelve neighboring cells remain closed during mixing and open only to route fluids into or out of the four mixing cells. Programs for mixing larger volumes (or different starting ratios of fluids) could be executed on the same device simply by including more cells into the recirculating loop. This highlights an important consequence of the programmability of a microfluidic cellular automaton: the device's program can be altered *in real time* to accommodate different analyte volumes, without altering the design of the device. For example, in the mixing program shown in Figure 6.7, additional cells could be filled with fluid in Step 1 until the entire volume of a sample is loaded on-chip. After mixing, the resulting large mixture could then be routed via a *group* of cells to subsequent operations which would, in turn, adapt in real time to operate upon the larger volume. In summary, this program is merely one of many different possible mixing programs—programs that operate on different volumes, mix at different speeds, distribute the resulting mixture in different directions and in different ratios, and so on—all of which operate interchangeably on the same device.

The ability to change its program in real time suggests another remarkable aspect of the microfluidic cellular automaton: it is *fault-tolerant*, able to identify and isolate

defective cells and still function correctly. Since the automaton's cells are fundamentally indistinguishable and interchangeable, an operation can easily be moved away from a stuck valve or clogged channel to empty cells elsewhere. This ability to "self-heal" would be especially useful in devices for extraterrestrial analysis, which must operate on other planets without human assistance [110].

By making a few small modifications to the mixing program shown in Figure 6.7, more-complex operations like serial dilutions can be performed in a microfluidic cellular automaton. Figure 6.8 depicts a program for performing an arbitrary number

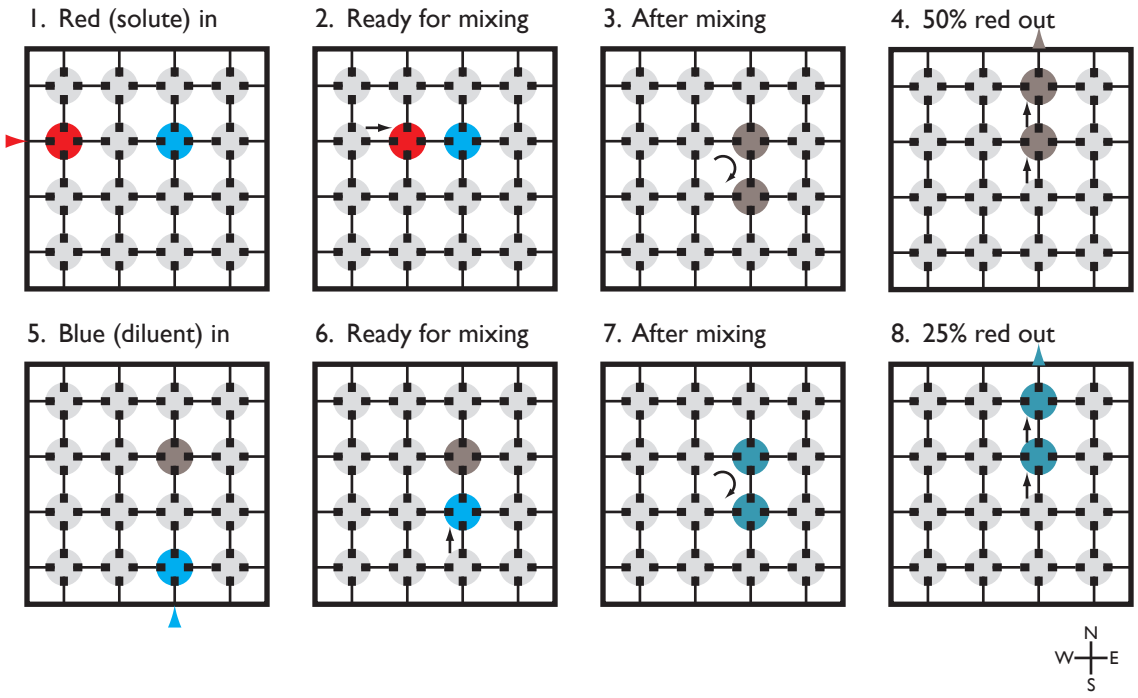


Figure 6.8: Sample program for performing "one-in-two" serial dilutions in a microfluidic cellular automaton.

of serial "one-in-two" dilutions in a cellular automaton. Steps 1 through 3 are an

abbreviated form of the mixing program shown in Figure 6.7. By Step 3, one cell's volume of solute (red) is mixed with one cell's volume of diluent (blue) to form two cells containing a 50% solution of solute in diluent. In Step 4, half of this mixture (a single cell's volume) is routed north for further processing or analysis, and a fresh cell's volume of fluid is routed into the mixing structure in Steps 5 and 6. After a second round of mixing in Step 7, the two cells contain a 25% solution of solute in diluent. Half of this mixture is routed north in Step 8, and Steps 5 through 8 are repeated as many times as necessary, routing a cell's volume of diluted solution north with every fourth step.

An important aspect of any microfluidic structure for serial dilution is the ability to specify the dilution ratio (the carryover fraction) between serial steps. In existing devices for automated serial dilution [135], the dilution ratio is fixed for a given device, and changing the ratio requires the design and fabrication of a new device. For a microfluidic cellular automaton running the dilution program in Figure 6.8, the dilution ratio can be increased or decreased at runtime by incorporating additional cells containing solute or diluent into the mixing loop in Steps 2 or 6.

Figure 6.9 presents a final sample program for a microfluidic cellular automaton, a program for replicating the behavior of the microfluidic processor used to perform single-nucleotide polymorphism-based DNA computations [136]. The general structure of the original processor is shown in Figure 6.9A. Three pump valves (red) are responsible for circulating fluid through six capture chambers (orange). Bus valves

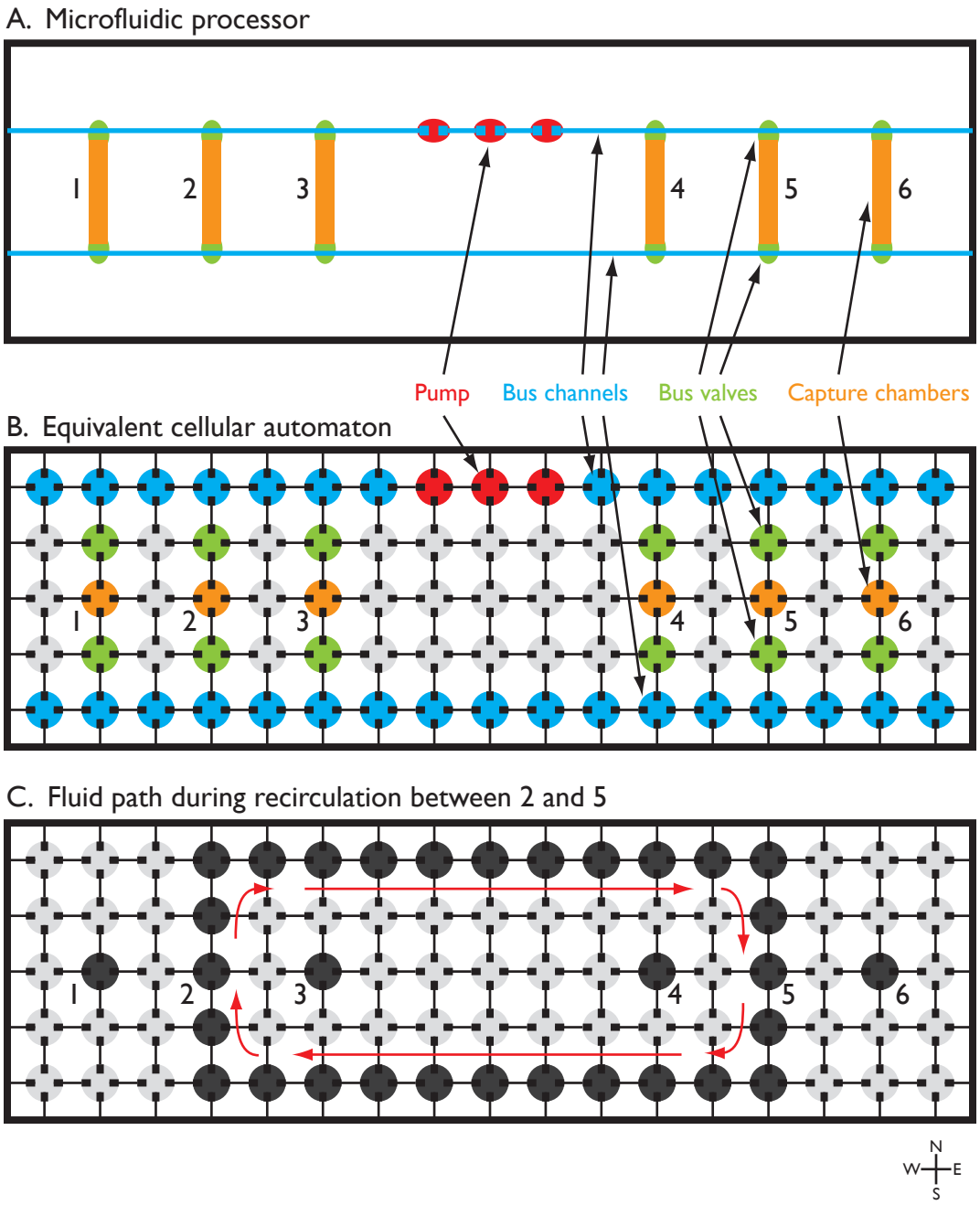


Figure 6.9: Traditional (A) and microfluidic cellular automaton (B) versions of the microfluidic processor used in [136]. (C) Open (dark gray) and closed (light gray) cells during recirculation between capture chamber cells 2 and 5. Red arrows indicate the clockwise path of fluid flow.

(green) are responsible for selecting which capture chambers are connected to the bus channels (blue) and incorporated into the recirculating fluid loop. The equivalent cellular automaton program is shown in 6.9B. Three neighboring cells (red) form a pump for circulating fluid through the other cells. Two rows of open cells (blue) function as bus channels. Six cells between the bus channels serve as capture chambers (orange); these capture chamber cells contain magnetic beads and must remain open. Each capture chamber is connected to the bus channel through cells that function as bus valves (green). By opening certain bus valve cells on the left and right sides of the device, selected capture chamber cells are incorporated into the recirculating loop. Figure 6.9C shows the open (dark gray) and closed (light gray) state of the cells during recirculation between capture chamber cells 2 and 5. Capture chamber cells that are not selected (chambers 1, 3, 4, and 6) remain open but are sealed off from the loop by closed bus valve cells.

The microfluidic cellular automaton is not limited to the parallel arrangement of capture chamber cells shown in Figure 6.9. Capture chamber cells can be arranged into any pattern, and fluid can be routed via any path through the selected capture chamber cells. The path (and the included capture chamber cells) can be changed without changing the bead contents of the capture chamber cells or redesigning the device. Flow through two or more capture chamber cells in parallel *and in series* is therefore possible. This could be especially useful in biological assays like haplotyping, in which serial flow through each biallelic SNP's capture chamber cells would be

preferable but is not possible on the current microfluidic processor [136].

6.6 Conclusions

This dissertation began with the question, *how do you compute in a microfluidic device?* Several answers to that question have been presented here. You can compute with valves that scale as well as transistors in conventional microprocessors. You can also compute with DNA, using the advantages of microfluidics to make single-base bits possible. You can even compute with circuits of pneumatic valves. These principles have found uses in lab-on-a-chip systems for pathogen and infectious disease detection [109], extraterrestrial amino acid analysis [110], pressure-injected electrophoretic separation [132], dielectrophoretic cell concentration [133], automated evolution of RNA catalysts [135], SNP-based DNA computation [136] nanoliter-scale Sanger DNA sequencing [134], demultiplexed latching valve control [140], and genetic haplotyping [145].

It is interesting to note the similarities that exist between the microfluidic devices used for these different applications. For example, recirculating on-chip fluid loops play a key role in both devices for automated ribozyme evolution and devices for DNA computing. Indeed, the device used for DNA computing is identical to the device currently being used for haplotyping. These similarities suggest that these devices are *evolving toward a common design*, a single processor that can support recirculating loops, affinity capture, PCR amplification, electrophoretic separation,

and many other important microfluidic operations. Clearly, there is not enough room on a microfluidic device for each of these operations to have a dedicated structure. Many of these operations could share a common, programmable structure that performs different operations (recirculating, mixing, affinity capture, PCR amplification, and so on) depending on how it is programmed and what inputs it receives. Even if operations like electrophoretic separation could never be integrated into the programmable structure and hence would always require a dedicated structure on-chip, these dedicated structures could still be interfaced to the shared structure, and the resulting device would still be more versatile than a single-purpose separation device.

The microfluidic cellular automaton described in this chapter is one possible design for this shared, programmable microfluidic processor. It is certainly not the only foreseeable design for a programmable microfluidic processor, and it may not the best design for the processor. But it is essential that the microfluidic community begin developing devices with more programmability and versatility than current single-purpose devices. There are over 600 different clinical tests in *Delmar's Guide to Laboratory and Diagnostic Tests* [160]; surely there is not enough manpower or funding in the world to develop a separate lab-on-a-chip device for each of them. To do so would be akin to having one computer for email, another computer for web browsing, a third computer for word processing, and so on. Computers advanced past this point fifty years ago. The time has come for microfluidic devices to do the same. It is the author's hope that the research presented in this dissertation will play

a role in the realization of this vision.

Bibliography

[1] M. Petkovsek, H. Wilf, and D. Zeilberger. *A = B*. A K Peters, Ltd., Wellesley, MA, 1996.

[2] G. Wood. *Edison's Eve: A Magical History of the Quest for Mechanical Life*. Anchor Books, New York, 2003.

[3] G. Ifrah. *The Universal History of Computing*. John Wiley and Sons, New York, 2001.

[4] A. W. J. G. Ord-Hume. *Pianola: The History of the Self-Playing Piano*. George Allen and Unwin, London, 1984.

[5] J. Perriault. *Éléments pour un dialogue avec l'informaticien*. Mouton, Paris, 1971 (translated in [3]).

[6] O. C. Faust. *A Treatise on the Construction, Repairing and Tuning of the Organ, Including Also the Reed Organ, the Orchestrelle and the Player-Piano*. Tuners Supply Company, Boston, 1949.

[7] P. Hagmann. *Das Welte-Mignon-Klavier, die Welte-Philharmonie-Orgel und die Anfänge der Reproduktion von Musik*. Peter Lang, Bern, 1984.

[8] J. von Neumann. The general and logical theory of automata. In A. H. Taub, editor, *John von Neumann: Collected Works*, volume 5, pages 288–328. Pergamon Press, New York, 1963.

[9] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[10] N. Forbes. *Imitation of Life: How Biology is Inspiring Computing*. MIT Press, Cambridge, Massachusetts, 2004.

[11] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[12] M. Gardner. Mathematical Games: The fantastic combinations of John Conway's new solitaire game 'Life'. *Scientific American*, 223(4):120–123, October 1970.

[13] S. Wolfram. *A New Kind of Science*. Wolfram Media Inc., Champaign, IL, 2002.

[14] M. Cook, P. W. K. Rothemund, and E. Winfree. Self-assembled circuit patterns. In J. Chen and J. Reif, editors, *DNA9: Ninth International Workshop on DNA-Based Computers, Lecture Notes in Computer Science 2943*, 2004.

[15] M. Conrad. Molecular computing. *Advances in Computers*, 31:235–324, 1990.

[16] C. Levinthal. How to fold graciously. In J. T. P. DeBrunner and E. Munck, editors, *Mossbauer Spectroscopy in Biological Systems: Proceedings of a meeting held at Allerton House, Monticello, Illinois*, pages 22–24. University of Illinois Press, 1969.

[17] R. R. Birge. Protein based computers. *Scientific American*, 272(3):90–95, March 1995.

[18] A. Adamatzky. *Molecular Computing*. MIT Press, Cambridge, Massachusetts, 2003.

[19] M. H. Garzon and R. J. Deaton. Biomolecular computing and programming. *IEEE Trans. on Evolutionary Computation*, 3(3):236–250, 1999.

[20] G. Paun, G. Rozenberg, and A. Salomaa. *DNA Computing: New Computing Paradigms*. Springer Verlag, Berlin, Heidelberg, New York, 1998.

[21] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[22] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, New York, 2001.

[23] L. M. Adleman. On constructing a molecular computer. In E. B. Baum R. J. Lipton, editor, *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–22. American Mathematical Society, 1996.

[24] R. J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.

[25] R. J. Lipton. Speeding up computations via molecular biology. In E. B. Baum R. J. Lipton, editor, *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 67–74. American Mathematical Society, 1996.

[26] P. D. Kaplan, G. Cecchi, and A. Libchaber. Molecular computation: Adleman's experiment repeated. Technical report, NEC Research Institute, 1995.

[27] J. Khodor and D. K. Gifford. The efficiency of sequence-specific separation of DNA mixtures for biological computing. In H. Rubin and D. H. Wood, editors, *DNA Based Computers III*, volume 48 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 39–46. American Mathematical Society, 1999.

[28] W. D. Smith. DNA computers in vitro and vivo. In E. B. Baum R. J. Lipton, editor, *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Math-*

ematics and Theoretical Computer Science, pages 121–186. American Mathematical Society, 1996.

[29] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296(5567):499–502, 2002.

[30] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for the satisfiability (SAT) problem: a survey. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–152. American Mathematical Society, 1997.

[31] D. Boneh, C. Dunworth, and R. J. Lipton. Breaking DES using a molecular computer. In E. B. Baum and R. J. Lipton, editors, *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 37–66. American Mathematical Society, 1996.

[32] L. M. Adleman, P. W. K. Rothemund, S. Roweis, and E. Winfree. On applying molecular computation to the data encryption standard. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

[33] J. C. Cox, D. S. Cohen, and A. D. Ellington. The complexities of DNA computation. *Trends in Biotechnology*, 17(4):151–154, Apr 1999.

[34] distributed.net, accessed March 15, 2006 at `http:\\www.distributed.net`.

[35] G. M. Whitesides, J. P. Mathias, and C. T. Seto. Molecular self-assembly and nanochemistry—a chemical strategy for the synthesis of nanostructures. *Science*, 254(5036):1312–1319, 1991.

[36] E. Winfree. On the computational power of DNA annealing and ligation. In E. B. Baum and R. J. Lipton, editors, *DNA Based Computers*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 199–210. American Mathematical Society, 1996.

[37] R. Holliday. Mechanism for gene conversion in fungi. *Genetical Research*, 5(2):282–FIX, 1964.

[38] T. J. Fu and N. C. Seeman. DNA double-crossover molecules. *Biochemistry*, 32(13):3211–3220, 1993.

[39] E. Winfree, X. Yang, and N. C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1999.

[40] E. Winfree, F. Lin, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394(6693):539–545, 1998.

[41] P. W. K. Rothemund, N. Papadakis, and E. Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biol.*, 2(12):2041–2053, 2004.

[42] R. Barish, P. W. K. Rothemund, and E. Winfree. Algorithmic self-assembly of a binary counter using DNA tiles. In A. Carbone, M. Daley, L. Kari, I. McQuillan, and N. Pierce, editors, *DNA Computing 11: Proceedings of the 11th International Meeting on DNA Computing, June 6–9*, London, Ontario, Canada, 2005.

[43] P. W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440:297–302, 2006.

[44] J.-T. Amenyo. Mesoscopic computer engineering: Automating DNA-based molecular computing via traditional practices of parallel computer architecture design. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1999.

[45] A. Gehani and J. Reif. Micro flow bio-molecular computation. *Biosystems*, 52(1-3):197–216, 1999.

[46] S. Roweis, E. Winfree, R. Burgoyne, N. V. Chelyapov, M. F. Goodman, P. W. K. Rothemund, and L. M. Adleman. A sticker based model for DNA computation.

In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1999.

[47] R. S. Braich, C. Johnson, P. W. K. Rothemund, D. Hwang, N. Chelyapov, and L. M. Adleman. Solution of a satisfiability problem on a gel-based DNA computer. In A. Condon and G. Rozenberg, editors, *DNA Computing: 6th International Workshop on DNA-Based Computers, Leiden, The Netherlands, June 13-17, 2000*, volume 2054 of *Lecture Notes in Computer Science*, pages 27–42. Springer Verlag, Berlin, Heidelberg, New York, 2001.

[48] M. Kenney, S. Ray, and T. C. Boles. Mutation typing using electrophoresis and gel-immobilized acrydite probes. *Biotechniques*, 25:516—521, 1998.

[49] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr. Good encodings for DNA-based solutions to combinatorial problems. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1999.

[50] J. S. McCaskill. Optically programming DNA computing in microflow reactors. *Biosystems*, 59(2):125–138, 2001.

[51] J. S. McCaskill, R. Penchovsky, M. Gohlke, J. Ackermann, and T. Rucker. Steady flow micro-reactor module for pipelined DNA computation. In A. Con-

don and G. Rozenberg, editors, *DNA Computing: 6th International Workshop on DNA-Based Computers, DNA 2000, Leiden, The Netherlands, June 13-17, 2000*, volume 2054 of *Lecture Notes in Computer Science*, pages 239–246. Springer Verlag, Berlin, Heidelberg, New York, 2001.

[52] D. van Noort, P. Wagler, and J. S. McCaskill. Hybrid poly(dimethylsiloxane)-silicon microreactors used for molecular computing. *Smart Materials and Structures*, 11:756–760, 2002.

[53] D. van Noort, Frank-Ulrich Gast, and J. S. McCaskill. DNA computing in microreactors. In N. Jonoska and N. C. Seeman, editors, *DNA Computing: 7th International Workshop on DNA-Based Computers, DNA7, Tampa, FL, USA, June 10-13, 2001. Lecture Notes in Computer Science*, volume 2340, pages 33–45. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[54] R. Penchovsky and J. S. McCaskill. Cascadable hybridization transfer of specific DNA between microreactor selection modules. In N. Jonoska and N. C. Seeman, editors, *DNA Computing: 7th International Workshop on DNA-Based Computers, DNA7, Tampa, FL, USA, June 10-13, 2001. Lecture Notes in Computer Science*, volume 2340, pages 46–56. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[55] V. A. Bloomfield, D. M. Crothers, and I. Tinoco, Jr. *Nucleic Acids: Structures, Properties, and Functions*. University Science Books, Sausalito, CA, 2000.

[56] D. van Noort and L. F. Landweber. Towards a re-programmable DNA computer. In J. Chen and J. Reif, editors, *DNA9: Ninth International Workshop on DNA-Based Computers, Lecture Notes in Computer Science*, volume 2943, pages 190–196, 2004.

[57] D. van Noort. A programmable molecular computer in microreactors. In C. Ferretti, G. Mauri, and C. Zandron, editors, *DNA10: 10th International Workshop on DNA-Based Computers, Lecture Notes in Computer Science*, volume 3348, pages 365–374, 2005.

[58] D. J. Harrison, K. Fluri, K. Seiler, Z. H. Fan, C. S. Effenhauser, and A. Manz. Micromachining a miniaturized capillary electrophoresis-based chemical-analysis system on a chip. *Science*, 261(5123):895–897, 1993.

[59] J. M. Ramsey and A. van den Berg, editors. *Proceedings of the MicroTAS 2001 Symposium, Monterey, CA, USA, October 21–25*, 2001.

[60] D. R. Reyes, D. Iossifidis, P. A. Auroux, and A. Manz. Micro total analysis systems. 1. Introduction, theory, and technology. *Anal. Chem.*, 74(12):2623–2636, 2002.

[61] H. J. Tian, A. Jaquins-Gerstl, N. Munro, M. Trucco, L. C. Brody, and J. P. Landers. Single-strand conformation polymorphism analysis by capillary and microchip electrophoresis: A fast, simple method for detection of common mutations in BRCA1 and BRCA2. *Genomics*, 63(1):25–34, 2000.

[62] M. C. Mitchell, V. Spikmans, A. Manz, and A. J. de Mello. Microchip-based synthesis and total analysis systems (μSYNTAS): Chemical microprocessing for generation and analysis of compound libraries. *J. Chem. Soc. Perk. T. 1*, 1(5):514–518, 2001.

[63] I. Medintz, W. W. Wong, L. Berti, L. Shiow, J. Tom, J. Scherer, G. Sensabaugh, and R. A. Mathies. High-performance multiplex SNP analysis of three hemochromatosis-related mutations with capillary array electrophoresis microplates. *Genome Res.*, 11(3):413–421, 2001.

[64] B. M. Paegel, C. A. Emrich, G. J. Weyemayer, J. R. Scherer, and R. A. Mathies. High throughput DNA sequencing with a microfabricated 96-lane capillary array electrophoresis bioprocessor. *Proc. Natl. Acad. Sci. USA*, 99(2):574–579, 2002.

[65] M. J. Powers, K. Domansky, M. R. Kaazempur-Mofrad, A. Kalezi, A. Capitano, A. Upadhyaya, P. Kurzawski, K. E. Wack, D. B. Stolz, R. Kamm, and L. G. Griffith. A microfabricated array bioreactor for perfused 3D liver culture. *Biotechnol. Bioeng.*, 78(3):257–269, 2002.

[66] L. D. Hutt, D. P. Glavin, J. L. Bada, and R. A. Mathies. Microfabricated capillary electrophoresis amino acid chirality analyzer for extraterrestrial exploration. *Anal. Chem.*, 71(18):4000–4006, 1999.

[67] R. C. Anderson, X. Su, G. J. Bogdan, and J. Fenton. A miniature integrated device for automated multistep genetic assays. *Nucleic Acids Res.*, 28:e60, 2000.

[68] E. T. Lagally, C. A. Emrich, and R. A. Mathies. Fully integrated PCR-capillary electrophoresis microsystem for DNA analysis. *Lab Chip*, 1(2):102–107, 2001.

[69] L. Bousse, S. Mouradian, A. Minalla, H. Yee, K. Williams, and R. Dubrow. Protein sizing on a microchip. *Anal. Chem.*, 73(6):1207–1212, 2001.

[70] M. T. Taylor, P. Belgrader, R Joshi, G. A. Kintz, and M. A. Northrup. Fully automated sample preparation for pathogen detection in a microfluidic cassette. In *Proceedings of the Micro Total Analysis Systems*, pages 670–672, Monterey, CA, USA, 2001.

[71] H. T. G. van Lintel, F. C. M. van de Pol, and S. Bouwstra. A piezoelectric micropump based on micromachining of silicon. *Sensor. Actuator.*, 15(2):153–167, 1988.

[72] T. T. Veenstra, J. W. Berenschot, J. G. E. Gardeniers, R. G. P. Sanders, M. C. Elwenspoek, and A. van den Berg. Use of selective anodic bonding to create micropump chambers with virtually no dead volume. *J. Electrochem. Soc.*, 148(2):G68–G72, 2001.

[73] D. L. Smith. *Thin-film Deposition: Principles and Practice*. McGraw-Hill, New York, 1995.

[74] C. Vieider, O. Ohman, and H. Elderstig. A pneumatically actuated micro valve with a silicone rubber membrane for integration with fluid-handling systems. In *Proceedings of the Eighth International Conference on Solid-State Sensors and Actuators, and Eurosensors IX*, pages 284–286, Stockholm, Sweden, 1995.

[75] L. Bousse, E. Dijkstra, and O. Guenat. High-density arrays of valves and interconnects for liquid switching. In *Proceedings of the Solid-State Sensor and Actuator Workshop*, pages 272–275, Hilton Head Island, SC, USA, 1996.

[76] T. Ohori, S. Shoji, K. Miura, and A. Yotsumoto. Partly disposable three-way microvalve for a medical micro total analysis system (μTAS). *Sensor. Actuator. A-Phys.*, 64(1):57–62, 1998.

[77] X. Yang, C. Grosjean, Y. C. Tai, and C. M. Ho. A MEMS thermopneumatic silicone rubber membrane valve. *Sensor. Actuator. A-Phys.*, 64(1):101–108, 1998.

[78] D. Baechi, R. Buser, and J. Dual. A high density microchannel network with integrated valves and photodiodes. *Sensor. Actuator. A-Phys.*, 95(2-3):77–83, 2002.

[79] S. Sjolander and C. Urbaniczky. Integrated fluid handling-system for biomolecular interaction analysis. *Anal. Chem.*, 63(20):2338–2345, 1991.

[80] S. Bohm, W. Olthuis, and P. Bergveld. A plastic micropump constructed with

conventional techniques and materials. *Sensor. Actuator. A-Phys.*, 77(3):223–228, 1999.

[81] W. K. Schomburg, R. Ahrens, W. Bacher, J. Martin, and V. Saile. AMANDA: Surface micromachining, molding, and diaphragm transfer. *Sensor. Actuator. A-Phys.*, 76(1-3):343–348, 1999.

[82] Y. N. Xia and G. M. Whitesides. Soft lithography. *Ann. Rev. Mater. Sci.*, 28:153–184, 1998.

[83] K. Hosokawa and R. Maeda. A pneumatically-actuated three-way microvalve fabricated with polydimethylsiloxane using the membrane transfer technique. *J. Micromech. Microeng.*, 10(3):415–420, 2000.

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

[85] X. Q. Ren, M. Bachman, C. Sims, G. P. Li, and N. Allbritton. Electroosmotic properties of microfluidic channels composed of poly(dimethylsiloxane). *J. Chromatogr B.*, 762(2):117–125, 2001.

[86] S. W. Hu, X. Q. Ren, M. Bachman, C. E. Sims, G. P. Li, and N. Allbritton. Surface modification of poly(dimethylsiloxane) microfluidic devices by ultraviolet polymer grafting. *Anal. Chem.*, 74(16):4117–4123, 2002.

[87] C. S. Effenhauser, G. J. M. Bruin, A. Paulus, and M. Ehrat. Integrated capillary electrophoresis on flexible silicone microdevices: Analysis of DNA restriction fragments and detection of single DNA molecules on microchips. *Anal. Chem.*, 69(17):3451–3457, 1997.

[88] J. W. Hong, T. Fujii, M. Seki, T. Yamamoto, and I. Endo. Integration of gene amplification and capillary gel electrophoresis on a polydimethylsiloxane-glass hybrid microchip. *Electrophoresis*, 22(2):328–333, 2001.

[89] D. J. Harrison, K. Fluri, N. Chiem, T. Tang, and Z. H. Fan. Micromachining chemical and biochemical analysis and reaction systems on glass substrates. *Sensor. Actuator. B-Chem.*, 33(1-3):105–109, 1996.

[90] J. P. Alarie, S. C. Jacobson, B. S. Broyles, T. E. McKnight, C. T. Culbertson, and J. M. Ramsey. Electroosmotically induced hudraulic pumping on microchips. In *Proceedings of the Micro Total Analysis Systems*, pages 131–132, Monterey, CA, USA, 2001.

[91] S. L. Zeng, C. H. Chen, J. G. Santiago, J. R. Chen, R. N. Zare, J. A. Tripp, F. Svec, and J. M. J. Frechet. Electroosmotic flow pumps with polymer frits. *Sensor. Actuator. B-Chem.*, 82(2-3):209–212, 2002.

[92] P. van der Voort and E. F. van Sant. Silylation of the silica surface: A review. *J. Liq. Chromatogr.*, 19(17-18):2723–2752, 1996.

[93] P. C. Simpson, A. T. Woolley, and R. A. Mathies. Microfabrication technology for the production of capillary array electrophoresis chips. *Biomed. Microdevices*, 1:7–26, 1998.

[94] E. T. Lagally, B. M. Paegel, and R. A. Mathies. Microfabrication technology for chemical and biochemical microprocessors. In *Proceedings of the 2000 Micro Total Analysis Systems conference*, pages 217–220, Enschede, The Netherlands, 2000.

[95] E. T. Lagally, P. C. Simpson, and R. A. Mathies. Monolithic integrated microfluidic DNA amplification and capillary electrophoresis analysis system. *Sensor. Actuator. B-Chem.*, 63(3):138–146, 2000.

[96] E. T. Lagally, I. Medintz, and R. A. Mathies. Single-molecule DNA amplification and analysis in an integrated microfluidic device. *Anal. Chem.*, 73(3):565–570, 2001.

[97] R. A. Mathies, E. T. Lagally, T. Kamei, W. H. Grover, C. N. Liu, and J. R. Scherer. Capillary array electrophoresis bioprocessors. In *Proceedings of the Solid-State Sensor, Actuator and Microsystems Workshop*, pages 112–117, Hilton Head Island, SC, USA, 2002.

[98] A. M. Skelley, F. J. Grunthaner, J. F. Bada, and R. A. Mathies. Mars Organic Detector III: A versatile instrument for detection of bio-organic signatures on

Mars. In P. M. Beauchamp G. H. Bearman, editor, *First Jet Propulsion Laboratory In Situ Instruments Workshop*, volume 4878. SPIE, Bellingham, WA, 2003.

[99] P. A. Auroux, D. Iossifidis, D. R. Reyes, and A. Manz. Micro total analysis systems. 2. analytical standard operations and applications. *Anal. Chem.*, 74(12):2637–2652, 2002.

[100] H. Andersson, W. van der Wijngaart, and G. Stemme. Micromachined filter-chamber array with passive valves for biochemical assays on beads. *Electrophoresis*, 22(2):249–257, 2001.

[101] Z. H. Fan, S. Mangru, R. Granzow, P. Heaney, W. Ho, Q. P. Dong, and R. Kumar. Dynamic DNA hybridization on a chip using paramagnetic beads. *Anal. Chem.*, 71(21):4851–4859, 1999.

[102] J. W. Hong, V. Studer, G. Hang, W. F. Anderson, and S. R. Quake. A nanoliter-scale nucleic acid processor with parallel architecture. *Nature Biotech.*, 22(4):435–439, 2004.

[103] A. B. Jemere, R. D. Oleschuk, F. Ouchen, F. Fajuyigbe, and D. J. Harrison. An integrated solid-phase extraction system for sub-picomolar detection. *Electrophoresis*, 23(20):3537–3544, 2002.

[104] G. H. Seong, W. Zhan, and R. M. Crooks. Fabrication of microchambers defined

by photopolymerized hydrogels and weirs within microfluidic systems: Application to DNA hybridization. *Anal. Chem.*, 74(14):3372–3377, 2002.

[105] E. Verpoorte. Beads and chips: New recipes for analysis. *Lab Chip*, 3(4):60N–68N, 2003.

[106] G. C. Kennedy, H. Matsuzaki, S. L. Dong, W. M. Liu, J. Huang, G. Y. Liu, X. Xu, M. Q. Cao, W. W. Chen, J. Zhang, W. W. Liu, G. Yang, X. J. Di, T. Ryder, Z. J. He, U. Surti, M. S. Phillips, M. T. Boyce-Jacino, S. P. A. Fodor, and K. W. Jones. Large-scale genotyping of complex DNA. *Nature Biotech.*, 21(10):1233–1237, 2003.

[107] W. H. Grover, A. M. Skelley, C. N. Liu, E. T. Lagally, and R. A. Mathies. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. *Sensor. Actuator. B-Chem.*, 89(3):315–323, 2003.

[108] B. M. Paegel, R. G. Blazej, and R. A. Mathies. Microfluidic devices for DNA sequencing: Sample preparation and electrophoretic analysis. *Curr. Opin. Biotech.*, 14(1):42–50, 2003.

[109] E. T. Lagally, J. R. Scherer, R. G. Blazej, N. M. Toriello, B. A. Diep, M. Ramchandani, G. F. Sensabaugh, L. W. Riley, and R. A. Mathies. Integrated portable genetic analysis microsystem for pathogen/infectious disease detection. *Anal. Chem.*, 76(11):3162–3170, 2004.

[110] A. M. Skelley, J. R. Scherer, A. D. Aubrey, W. H. Grover, R. H. C. Ivester, P. Ehrenfreund, F. J. Grunthaner, J. L. Bada, and R. A. Mathies. Development and evaluation of a microdevice for amino acid biomarker detection and analysis on Mars. *Proc. Natl. Acad. Sci. USA*, 102(4):1041–1046, 2005.

[111] P. C. Simpson, D. Roach, A. T. Woolley, T. Thorsen, R. Johnston, G. F. Sensabaugh, and R. A. Mathies. High-throughput genetic analysis using microfabricated 96-sample capillary array electrophoresis microplates. *Proc. Natl. Acad. Sci. USA*, 95(5):2256–2261, 1998.

[112] M. S. Livstone and L. F. Landweber. Mathematical considerations in the design of microreactor-based DNA computers. *DNA Computing*, 2943:180–189, 2004.

[113] N. Peyret, P. A. Seneviratne, H. T. Allawi, and J. SantaLucia. Nearest-neighbor thermodynamics and NMR of DNA sequences with internal AA, CC, GG, and TT mismatches. *Biochemistry*, 38(12):3468–3477, 1999.

[114] M. Grompe, D. M. Muzny, and C. T. Caskey. Scanning detection of mutations in human ornithine transcarbamoylase by chemical mismatch cleavage. *Proc. Natl. Acad. Sci. USA*, 86(15):5888–5892, 1989.

[115] P. C. Solaro, K. Birkenkamp, P. Pfeiffer, and B. Kemper. Endonuclease-VII of phage-T4 triggers mismatch correction in vitro. *J. Mol. Bio.*, 230(3):868–877, 1993.

[116] R. G. Blazej, B. M. Paegel, and R. A. Mathies. Polymorphism ratio sequencing: A new approach for single nucleotide polymorphism discovery and genotyping. *Genome Res.*, 13(2):287–293, 2003.

[117] N. Patil, A. J. Berno, D. A. Hinds, W. A. Barrett, J. M. Doshi, C. R. Hacker, C. R. Kautzer, D. H. Lee, C. Marjoribanks, D. P. McDonough, B. T. N. Nguyen, M. C. Norris, J. B. Sheehan, N. P. Shen, D. Stern, R. P. Stokowski, D. J. Thomas, M. O. Trulson, K. R. Vyas, K. A. Frazer, S. P. A. Fodor, and D. R. Cox. Blocks of limited haplotype diversity revealed by high-resolution scanning of human chromosome 21. *Science*, 294(5547):1719–1723, 2001.

[118] M. Jobs, W. M. Howell, L. Stromqvist, T. Mayr, and A. J. Brookes. DASH-2: Flexible, low-cost, and high-throughput SNP genotyping by dynamic allele-specific hybridization on membrane arrays. *Genome Res.*, 13(5):916–924, 2003.

[119] L. M. Wang, J. G. Hall, M. C. Lu, Q. H. Liu, and L. M. Smith. A DNA computing readout operation based on structure-specific cleavage. *Nature Biotech.*, 19(11):1053–1059, 2001.

[120] Q. H. Liu, L. M. Wang, A. G. Frutos, A. E. Condon, R. M. Corn, and L. M. Smith. DNA computing on surfaces. *Nature*, 403(6766):175–179, 2000.

[121] C. A. Emrich, H. J. Tian, I. L. Medintz, and R. A. Mathies. Microfabricated 384-lane capillary array electrophoresis bioanalyzer for ultrahigh-throughput genetic analysis. *Anal. Chem.*, 74(19):5076–5083, 2002.

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

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

[124] C. H. Roth, Jr. *Fundamentals of logic design*. West Publishing Company, 1985.

[125] B. Wagner, H. J. Quenzer, S. Hoerschelmann, T. Lisec, and M. Juerss. Bistable microvalve with pneumatically coupled membranes. In *Proceedings of the Ninth Annual IEEE International Workshop on Micro Electro Mechanical Systems*, pages 384–388, 1996.

[126] C. Goll, W. Bacher, B. Bustgens, D. Maas, W. Menz, and W. K. Schomburg. Microvalves with bistable buckled polymer diaphragms. *J. Micromech. Microeng.*, 6(1):77–79, 1996.

[127] J. A. Potkay and K. D. Wise. An electrostatically latching thermopneumatic microvalve with closed-loop position sensing. In *Proceedings of the 18th IEEE International Conference on Micro Electro Mechanical Systems*, pages 415–418, 2005.

[128] J. Y. Pan, D. VerLee, and M. Mehregany. Latched valve manifolds for efficient control of pneumatically actuated valve arrays. In *Proceedings of the IEEE*

International Conference on Solid State Sensors and Actuators, pages 817–820, 1997.

[129] S. Bohm, G. J. Burger, M. T. Korthorst, and F. Roseboom. A micromachined silicon valve driven by a miniature bi-stable electro-magnetic actuator. *Sensor. Actuator. A-Phys.*, 80(1):77–83, 2000.

[130] M. Capanu, J. G. Boyd, and P. J. Hesketh. Design, fabrication, and testing of a bistable electromagnetically actuated microvalve. *J. Microelectromech. S.*, 9(2):181–189, 2000.

[131] R. Pal, M. Yang, B. N. Johnson, D. T. Burke, and M. A. Burns. Phase change microvalve for integrated devices. *Anal. Chem.*, 76(13):3740–3748, 2004.

[132] J. M. Karlinsey, J. Monahan, D. J. Marchiarullo, J. P. Ferrance, and J. P. Landers. Pressure injection on a valved microdevice for electrophoretic analysis of submicroliter samples. *Anal. Chem.*, 77(11):3637–3643, 2005.

[133] E. T. Lagally, S. H. Lee, and H. T. Soh. Integrated microsystem for dielectrophoretic cell concentration and genetic detection. *Lab Chip*, 5(10):1053–1058, 2005.

[134] R. G. Blazej, P. Kumaresan, and R. A. Mathies. Microfabricated bioprocessor for integrated nanoliter-scale Sanger DNA sequencing. *Proc. Natl. Acad. Sci. USA*, in press.

[135] B. M. Paegel and G. F. Joyce. Microfluidic serial transfer circuit: Automated evolution of RNA catalysts. In K. F. Jensen, J. Han, D. J. Harrison, and J. Voldman, editors, *Proceedings of the MicroTAS 2005 Conference, Boston, MA, USA, October 9–13*, pages 28–30, 2005.

[136] W. H. Grover and R. A. Mathies. An integrated microfluidic processor for single nucleotide polymorphism-based DNA computing. *Lab Chip*, 5:1033–1040, 2005.

[137] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[138] D. M. Fleetwood, P. S. Winokur, and P. E. Dodd. An overview of radiation effects on electronics in the space telecommunications environment. *Microelectron. Reliab.*, 40:17–26, 2000.

[139] D. R. Lide and H. P. R. Frederikse, editors. *The CRC Handbook of Chemistry and Physics*. Chemical Rubber Publishing Company, Boca Raton, FL, 77th edition, 1996.

[140] W. H. Grover, R. H. C. Ivester, E. C. Jensen, and R. A. Mathies. Development and multiplexed control of latching pneumatic valves using microfluidic logical structures. *Lab Chip*, 6:623–631, 2006.

[141] R. Tran. In preparation.

[142] S. Hjerten. High-performance electrophoresis: Elimination of electroendosmosis and solute adsorption. *J. Chromatogr.*, 347:191–198, 1985.

[143] F. Svec and J. M. J. Frechet. Molded rigid monolithic porous polymers: An inexpensive, efficient, and versatile alternative to beads for the design of materials for numerous applications. *Industrial and Engineering Chemistry Research*, 38(1):34–48, Jan 1999.

[144] L. F. Landweber. DNA^{2}DNA computations: A potential "killer app"? In H. Rubin and D. H. Wood, editors, *DNA Based Computers III*, volume 48 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 59–68. American Mathematical Society, 1999.

[145] E. C. Jensen. In preparation.

[146] C. Jurinke, D. van den Boom, V. Collazo, A. Luchow, A. Jacob, and H. Koster. Recovery of nucleic acids from immobilized biotin-streptavidin complexes using ammonium hydroxide and applications in maldi-tof mass spectrometry. *Analytical Chemistry*, 69(5):904–910, 1997.

[147] J. D. Hurley, L. J. Engle, J. T. Davis, A. M. Welsh, and J. E. Landers. A simple, bead-based approach for multi-SNP molecular haplotyping. *Nucleic Acids Research*, 32(22):e186, 2004.

[148] A. D. Ellington and J. W. Szostak. Invitro selection of RNA molecules that bind specific ligands. *Nature*, 346(6287):818–822, 1990.

[149] G. F. Joyce. Amplification, mutation and selection of catalytic RNA. *GENE*, 82(1):83–87, 1989.

[150] C. Tuerk and L. Gold. Systematic evolution of ligands by exponential enrichment—RNA ligands to bacteriophage-T4 DNA-polymerase. *Science*, 249(4968):505–510, 1990.

[151] L. F. Landweber. RNA based computing: Some examples from RNA catalysis and RNA editing. In L. Landweber and E. Baum, editors, *DNA Based Computers II*, volume 44 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pages 181–189. American Mathematical Society, 1999.

[152] A. R. Cukras, D. Faulhammer, R. J. Lipton, and L. F. Landweber. Chess games: a model for rna based computation. *BioSystems*, 52(1-3):35–45, 1999.

[153] D. Faulhammer, A. R. Cukras, R. J. Lipton, and L. F. Landweber. Molecular computation: RNA solutions to chess problems. *Proc. Natl. Acad. Sci. USA*, 97(4):1385–1389, 2000.

[154] F. G. Li and G. D. Stormo. Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics*, 17(11):1067–1076, 2001.

[155] R. P. Feynman. There's plenty of room at the bottom. *Engineering and Science*, 23:22–36, 1960.

[156] K. Somei, S. Kaneda, T. Fujii, and S. Murata. A microfluidic device for DNA tile self-assembly. In A. Carbone, M. Daley, L. Kari, I. McQuillan, and N. Pierce, editors, *DNA Computing 11: Proceedings of the 11th International Meeting on DNA Computing, June 6–9*, London, Ontario, Canada, 2005.

[157] R. Schulman and E. Winfree. Programmable control of nucleation for algorithmic self-assembly. In C. Ferretti, G. Mauri, and C. Zandron, editors, *DNA10: 10th International Workshop on DNA-Based Computers, Lecture Notes in Computer Science 3348*, pages 319–328, 2005.

[158] P. Wagler, U. Tangen, T. Maeke, H. P. Mathis, and J. S. McCaskill. Microfabrication of a biomodule composed of microfluidics and digitally controlled microelectrodes for processing biomolecules. *Smart Materials and Structures*, 12:757–762, 2003.

[159] M. G. Pollack, R. B. Fair, and A. D. Shenderov. Electrowetting-based actuation of liquid droplets for microfluidic applications. *Applied Physics Letters*, 77(11):1725–1726, 2000.

[160] R. Daniels. *Delmar's Guide to Laboratory and Diagnostic Tests*. Delmar, Albany, NY, 2002.

Appendix A

OCW language reference

The two main versions of OCW (see Section 5.9) have slightly different commands. The original Linux version for valve control via parallel port interface (used in [136] and [140]) includes port address setting commands that are unnecessary in the LabVIEW version. Likewise, the newer LabVIEW version for control via National Instruments hardware (used in [145]) includes features for complex OCW programs that are unavailable in the original Linux version. In the following language references, all commands are assumed valid for both versions of OCW, with exceptions marked as *Linux only* or *LabVIEW only*. This reference was typeset using the conventions shown in Table A.1:

Font	Meaning
`fixed width`	OCW keyword
italics	User-specified variable
sans serif	Keyboard or on-screen control

Table A.1: Typographical conventions for the OCW language reference.

`o`n Open valve n.

Available valve numbers are 0 through $8n-1$ where n is the number of parallel ports (Linux only) or digital output ports (LabVIEW only) available on the system. In the Linux version of OCW, valves 0 through 7 are assigned to the first port address set, valves 8 through 15 are assigned to the second port address set, and so on (see the `a` command below). In the LabVIEW version of OCW, the order of the digital output ports is set in on the front panel of the VI. If the valve is already open, it will remain open.

`c`n Close valve n.

Available valve numbers are the same as for the `o` command. If the valve is already closed, it will remain closed.

`w`t Wait t milliseconds.

The accuracy of the wait command is system-dependent; extremely short waits (< 100 ms) may be inaccurate.

/comment Comment line.

If the forward-slash `/` is placed at the beginning of a line, all remaining text on that line is ignored. In addition, comments within executed functions (either `main` or a subroutine) are printed on the screen during execution. Comments placed before a `stop` command can therefore be used to tell the operator what to do during that stop, e.g. "`/`*Add more buffer*".

`stop` Stop program execution for operator intervention.

Execution stops until the operator presses ENTER (Linux only) or clicks Resume (LabVIEW only).

`main` Begin `main` function.

Every OCW program must have a `main` function. The `main` function cannot be repeated arbitrarily like subroutines can. Instead, place the contents of the `main` function in a subroutine and `call` that subroutine from `main` using repeats (see `call` below).

`end` End `main` program or a subroutine.

The `main` function and each subroutine are terminated by `end` commands. The code lying between `main` and the main function's `end` is the `main` function; code between the declaration of a subroutine (see below) and that subroutine's `end` is that subroutine's code.

sfname Begin definition of the subroutine named *sfname*.

Any line containing a single word that isn't another OCW function is assumed to be the declaration of a subroutine. The name of the subroutine marks the beginning of the subroutine's code, and the corresponding `end` command marks the end of the code. Underscores can be included in the subroutine names for legibility, as in *pump_forward*.

`call` ***sfname*** *n* Call subroutine *sfname* *n* times, once if *n* is omitted.

When a subroutine is called, its code is executed as if the subroutine's code was actually inserted in the program at the `call` command. Adding an integer number *n* after the function name in the `call` command will repeat that subroutine's code *n* times; if the number is omitted the code is executed once. By specifying a excessively large number of repeats and pressing ENTER (Linux only) or clicking Escape (LabVIEW only) at runtime to escape the repeating subroutine, the number of repeats executed can be controlled at runtime by the operator (see below).

`include` ***filename.ocw*** Include contents of file *filename.ocw* **(LabVIEW only)**.

The `include` command allows a complex OCW program to span multiple source files. The contents of an `include`d file are executed when the `include` statement is encountered, as if they were part of the current file. Commonly-used subroutines like *pump_forward* or *close_all_valves* can be defined once in a sin-

gle, shared file and `included` into other files without having to redefining them, making the resulting OCW scripts much shorter and easier to understand.

`a`*n* Set next parallel port address to *n* **(Linux only)**.

The `a` command is called as many times as there are parallel ports to be used on the computer. The order in which the ports are called is used to determine which set of 8 valves is assigned to which parallel port. The first parallel port address set using `a` will control valves 0 through 7, the second port address set with `a` will control valves 8 through 15, and so on. The port address is set using the decimal equivalent of the (usually hexidecimal) parallel port address, as shown in Table A.2.

Port address (hexidecimal)	OCW code (decimal)
`0x3BC`	`a`*956*
`0x378`	`a`*888*
`0x278`	`a`*632*
`0x368`	`a`*872*
`0x268`	`a`*616*
`0x358`	`a`*856*
`0x258`	`a`*600*

Table A.2: Common hexidecimal parallel port addresses and corresponding decimal OCW `a` commands.

`armed` Activate parallel ports **(Linux only)**.

Without an `armed` command in the beginning of the file, the OCW code will execute but will not actually control the parallel ports ("debug mode"). Include the `armed` command to actually control the parallel ports.

`negate` Negate parallel port logic **(Linux only)**.

The OCW interpreter ordinarily assumes that opening a valve corresponds to energizing a solenoid valve (sending out a "high" voltage from the parallel port) and closing a valve corresponds to de-energizing a solenoid valve (sending out a "low" voltage). Depending on the type of solenoid valve used and the construction of the pneumatic control system, this assumption may need to be reversed. If the `negate` command is included in the beginning of an OCW file, the de-energized state of the solenoid valve will correspond to an *open* valve and the energized state will correspond to a *closed* valve.

Finally, a few commands may be typed or clicked while the OCW program is running. These commands affect the execution of the OCW program being run:

ENTER Resume execution following a `stop` command **(Linux only)**.

Resume Resume execution following a `stop` command **(LabVIEW only)**.

ENTER Escape the current repeating subroutine **(Linux only)**.

Escape Escape the current repeating subroutine **(LabVIEW only)**.

Pause Pause program execution **(LabVIEW only)**.

Appendix B

Sample OCW script

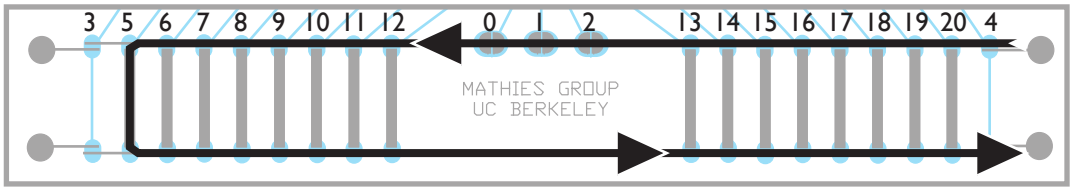


Figure B.1: Filling chamber A of the microfluidic processor used in [136]. Valves are labeled with the numbers of their pneumatic control lines.

```
/ "Load chamber A" program by Will Grover

/ The main program:
main
/ Make sure all valves start closed:
call close_all_valves
/ Load beads in input reservoir
/ Click "Proceed" when finished
stop
/ Open the right input/output valves and chamber A:
o4
o5
```

```
w1000
/ Call "pump_left" subroutine 1000 times (or until
/ user presses "Escape"):
call pump_left 10000
/ Graceful shutdown:
call close_all_valves
end

/ The close_all_valves subroutine:
close_all_valves
c0
c1
c2
...lines omitted...
c19
c20
end

/ The pump_left subroutine:
pump_left
o2
call pump_wait
c0
call pump_wait
o1
call pump_wait
c2
call pump_wait
o0
call pump_wait
c1
call pump_wait
end

/ The pump_wait subroutine:
pump_wait
w500
end
```

Appendix C

OCW source code for Linux

ocw.pl

```
#!/usr/bin/env perl
#OCW v0.91 for Linux by Will Grover

@in=<>;
$|=1;
my @state;

# The main program:  Call the "ocw" subroutine with
# the target "main" and the repeat count "1":
ocw(main,1);

# "key_ready" subroutine
# (c) 2006 by Tom Christiansen and Nathan Torkington
# Released into the public domain at
# http://perldoc.perl.org/perlfaq5.html
sub key_ready {
 my($rin, $nfd);
 vec($rin, fileno(STDIN), 1) = 1;
 return $nfd = select($rin,undef,undef,0);
}
```

```
# The recursive subroutine 'ocw' is passed two arguments,
# the name of the target OCW subroutine,
# and the number of times to repeat the target:
sub ocw {
 my $target=shift(@_);
 my $times=shift(@_);

 # The main FOR loop,
 # executed once for each repeat of the current target:
 for(my $time=0;$time<$times;$time++) {

  # If a key has been pressed to escape the current
  # repeating OCW subroutine, escape the main FOR loop:
  if(key_ready) {
   print "\a";
   if($times>1) {
    $input=<STDIN>;
   }
   last;
  }

  # Parse through each line in the OCW code
  # looking for either preamble instructions
  # (which are operated upon regardless of whether the
  # target subroutine has been found) or the
  # target subroutine (whose contents are also operated upon).
  # Code in non-target OCW subroutines is ignored.
  my $intarget=0;
  my @addresses;
  my $address=0;
  foreach (@in) {

   # Regular expression for the 'armed' preamble command.
   if(/^armed/) {
    $armed=1;
   }

   # Regular expression for the 'negate' preamble command.
   elsif(/^negate/) {
    $negate=1;
   }
```

```
# Regular expression for the port address-setting 'a' command.
elsif(/^a(\d+)/) {
 @addresses[$address]=$1;
 $address++;
}

# Regular expression for finding the start
# of the target OCW subroutine.
elsif(/^$target/) {
 $intarget=1;
}

# Regular expression for finding the end
# of the target OCW subroutine.
elsif(/^end/) {
 $intarget=0;
}

# Regular expression for comments within
# the target OCW subroutine.
elsif(/\//&&$intarget) {
 print "\n$_";
}

# Regular expression for calling sub-subroutines
# within the target OCW subroutine.
elsif(/^call (\w+) *(\d*)/&&$intarget==1) {

 # If the number of repeats is specified, recursively call
 # the "ocw" subroutine using the new target and repeats.
 if($2) {
  ocw($1,$2);
 }

 # Otherwise, recursively call the "ocw" subroutine
 # using the new target and "1" repeat.
 else {
  ocw($1,1);
 }
}

# Regular expression for "o" (open):
```

```
   elsif(/o(\d+)/&&$intarget) {
    print "o$1 ";
    $state[$1/8]=$state[$1/8]|(2**($1%8));
    if($armed==1) {
     if($negate==1) {
      system './linux', $addresses[$1/8], 255-$state[$1/8];
     }
     else {
      system './linux', $addresses[$1/8], $state[$1/8];
     }
    }
   }

   # Regular expression for "c" (close):
   elsif(/c(\d+)/&&$intarget) {
    print "c$1 ";
    $state[$1/8]=$state[$1/8]&~(2**($1%8));
    if($armed==1) {
     if($negate==1) {
      system './linux', $addresses[$1/8], 255-$state[$1/8];
     }
     else {
      system './linux', $addresses[$1/8], $state[$1/8];
     }
    }
   }

   # Regular expression for "w" (wait):
   elsif(/w(\d+)/&&$intarget) {
    print "w$1 ";
    select(undef,undef,undef,$1/1000);
   }
   elsif(/^stop/&&$intarget) {
    $input = <STDIN>;
   }
  }
 }
}
```

OCW.C

```
/* OCW v0.91 for Linux by Will Grover */

#include <stdio.h>
#include <sys/io.h>

int main(int argc, char **argv) {
 ioperm(atoi(argv[1]),1,1);
 outb(atoi(argv[2]),atoi(argv[1]));
 return 0;
}
```