Design and Verification Tools for Continuous Fluid Flow-based Microfluidic Devices

Jeffrey McDaniel, Auralila Baez, Brian Crites, Aditya Tammewar, Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521

Abstract – This paper describes an integrated design, verification, and simulation environment for programmable microfluidic devices called laboratories-on-chip (LoCs). Today's LoCs are architected and laid out by hand, which is time-consuming, tedious, and error-prone. To increase designer productivity, this paper introduces a Microfluidic Hardware Design Language (MHDL) for LoC specification, along with software tools to assist LoC designers verify the correctness of their specifications and estimate their performance.

I. Introduction

Laboratory-on-chip (LoC) technology, which integrates laboratory functions into integrated devices, offers the potential to revolutionize the fields of chemistry, biochemistry, and bioengineering through programmability, automation, and miniaturization. The advent of *microfluidic* technology, which addresses the precise control and manipulation of micro-scale liquid quantities, coupled with advances in microfabrication techniques, has enabled the development of low-cost LoCs.

The burgeoning industry for microfluidic technology and LoCs is still in its infancy. At present, human designers, who are not device experts, specify LoCs using software such as AutoCAD; this approach provides no separation between architecture and physical layout, and is time-consuming, tedious, and error-prone. Motivated by *electronic design automation (EDA)* processes in the semiconductor industry, this paper introduces a *Microfluidic Hardware Design Language (MHDL)* for LoC specification, along with software that performs verification and performance estimation. Once the designer is satisfied, he or she can obtain a netlist from the MHDL compiler and lay out the LoC in AutoCAD.

Introducing EDA to the microfluidics industry will enable specialization. At present, the user base of LoCs, namely chemists, biologists, and bioengineers, are also the designers, which introduces insurmountable inefficiencies: when an LoC is needed to perform an experiment, the user—whose research objective is in the biological sciences—must put that objective on hold in order to design and fabricate a usable LoC that can enable the research. Without EDA support, the microfluidics industry will remain paralyzed in a state akin to the semiconductor industry of the early 1960's; this paper takes an initial step toward design automation for LoCs.

A. Technology Overview

At present, there are a variety of microfluidic technologies that are vying to gain traction, both experimentally in laboratories, and commercially in industry. The tool we are developing has been designed to work with any technology that is properly defined.

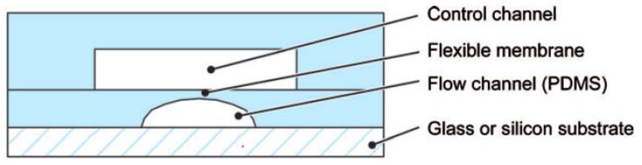


Fig. 1. A programmable microvalve [14]; applying pressure through the control channel closes the microvalve.

The devices we have focused on use programmable microvalves, as shown in Fig. 1., as the method of control [4, 10, 14]. These microvalves are fabricated in silicone polymers such as polydimethylsiloxane (PDMS) using soft lithography [22] and very large-scale integration of these valves has been achieved [4, 10]. Programmable microvalve technology has been used to build components such as mixers [21], memories [21], and multiplexers [9]; larger devices have also been constructed, including, but not limited to an LoC for influenza detection [16] and programmable arrays that can be configured to perform a variety of functions [7, 12].

Microvalve control is provided by external solenoids; control inputs are quite large compared to microvalves, which creates many challenges for LoC design. One approach is to limit the number of integrated microvalves in the device to one-per-control input [12], or to find situations where one control input can control multiple valves whose actuations are mutually independent [2]. A second approach is introduce a demultiplexer, that enables *n* control inputs to independently control 2^n microvalves [7, 9]; however, the area overhead of the demultiplexer is significant: in one case, the demultiplexer is several times larger than the rest of the LoC [7]. A third approach, whose feasibility has been demonstrated within the past year, is to implement a state machine controller using programmable microvalve logic [6, 15]. In principle, our toolflow can support al of these control styles, as long as the user specifies them correctly in MHDL.

B. Motivation

Our motivation to design a fluidic HDL is based on the observation that many LoCs can be viewed as collections of components (e.g., valves, mixers, fluid memories, etc.) connected by fluid routing channels; such a representation is conceptually similar to a netlist representation of a circuit. This programmable interface has some principle similarities to the machine language of a processor; the best analogue is UC Irvine's No Instruction Set Computing (NISC) [8], which foregoes a formal instruction set and instead provides direct access to the control interface of each pipeline component.

C. Overview

This paper introduces the first steps towards a tool chain for LoC specification, debugging and functional verification, and high-level performance analysis. The biological protocol executed by an LoC is called an *assay*. The tool chain, shown in Fig. 2, abstracts the assay specification away from the technology of the LoC that will execute it. Likewise, the LoC specification using MHDL is abstracted away from its physical layout. These abstractions will serve to enable specialization among LoC designers and users.

The user of an LoC, such as a biologist or chemist, should focus on specifying assays, just as a software engineer specifies algorithms with programming languages; software engineers are not digital designers, and do not need to design the platforms that runs their applications. Similarly, the LoC designer specifies the architecture using MHDL, in a manner that is independent from the layout.

Lastly, device engineers, who are component experts, but not LoC architects, will specify the entity libraries, which contain components that may be physically laid out a-priori; this is similar to the way the semiconductor industry employs reusable components such as standard cells or pre-placed-and-routed IP blocks [13]. Files describing entity libraries can be created without any specific LoC or assay in mind.

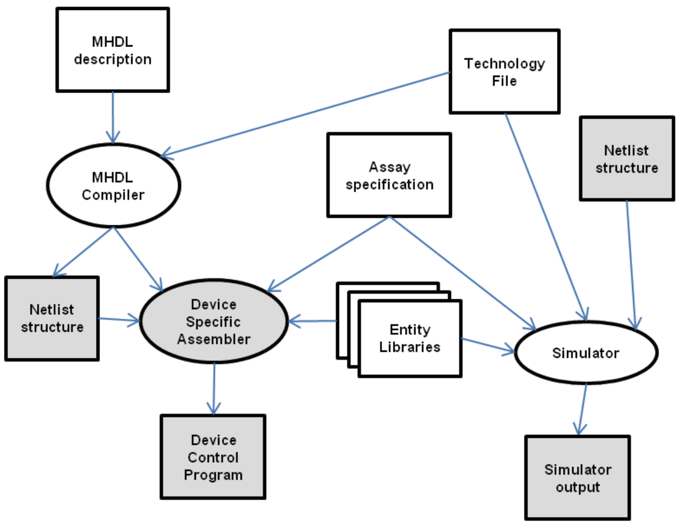


Fig. 2. A flow diagram of the software tool chain presented in this paper: The white boxes are user-created text files, which are read by the various tools. The gray boxes are text files that are generated by the tools, or used as input to later parts of the flow. The white ovals are static software modules that we create. The gray ovals are software modules that are generated by other modules automatically. The netlist structure on the right hand side is the same one that is generated on the left hand side; it has been replicated for clarity.

II. File Formats

A. Microfluidic HDL

Our MHDL syntax, shown in Fig. 3, is based on VHDL and has been designed to be easily extensible to deal with the creation of new components as well as the development of new technologies. The flexibility of the compiler comes from the use of library files during the compilation process so the MHDL syntax does not need to change when new LoC components and technologies are introduced.

The MHDL file starts by naming the current chip design, shown in the *Design* rule in Fig. 3. The *entity types* are then declared in the *entities* section, followed by *components* and *connections* between components. The user may optionally declare *intermediate lines* to be used, if desired.

The compiler needs to know whether each component is *active* or *passive*, so that it can represent it correctly internally; an active component is declared *with control* while a passive component is declared *without control*. Entities are declared with the same name as the library file associated with them, but without the file extension. The *components* rule declares each component and requires an entity type to be specified. The syntax allows the user to declare one or more components having the same entity type within a single statement.

The *connections* rule defines routing channels between components. Specific *lines* can be optionally declared for clarity. Each connection is directional with a single *source* component that injects fluid into it; this component is specified first, followed by a list of *sinks*, which receive fluid. Multiple directional connections with difference sources can be used in lieu of a bi-directional channel.

The *blocks* keyword enables the user to provide estimates of the length of a fluid channel to enable simulation of fluid transfer latencies; the exact length is not known until the physical layout of the LoC has been finalized.

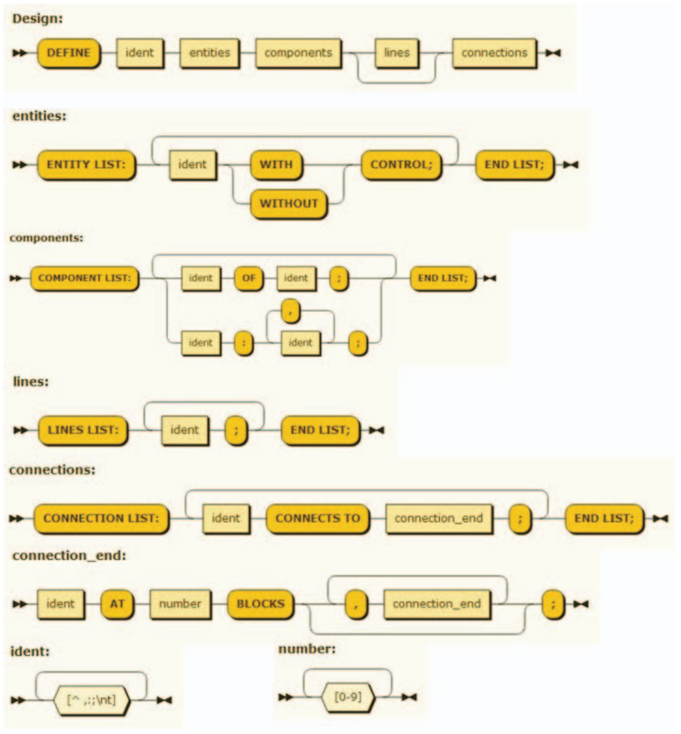


Fig. 3. A syntax diagram for MHDL. This language provides support for defining the entities, declaring the components, and describing the interconnection between the components.

Usage of the *blocks* keyword will provide some guidance to physical layout tools, which we plan to develop in the future; for now, these issues are left open for future work.

B. Entities and Library Files

LoCs are not created from entirely unique components; there can be multiple components with the same functionality, which only need to be defined once. The components that the designer can utilize are the entities for which a library file is defined. As entities themselves are technology-dependent, so is each library file. Therefore a single entity type may have multiple library files for each technology in which it is implemented. The library file describes the operations that the entity supports, as well as the entity's control interface for the specific technology. A component engineer who creates a new entity type can create a corresponding library file describing its functionality. This abstracts the high-level chip design away from the underlying hardware development, which increases specialization and designer productivity.

C. Device-specific Machine Language

Each active component in LoC will have one or more control signals connected to it. The control signals may be different, depending on the technology that implements the component, the control scheme (as discussed in Section I.A), and whether the control is driven by pressure or electrical actuation (a technology-dependent issue).

The control signals can be viewed as a vector, whose values are varied at each time-step as the LoC "program" executes. For electrical signals, the control vector is Boolean-valued, whereas, for pressure-based signals, the control vector is real-valued, where each scalar in the vector is the pressure applied to the control input. Conceptually, each vector can be viewed as a machine language instruction, as it completely defines the instantaneous behavior of the active components that comprise the LoC. Thus, a sequence of control vectors can be viewed as a program. This enables compilation: a compiler targeting the LoC is a software program that converts an assay specification [3, 19, 20] into a sequence of control vectors that execute the assay on the LoC.

III. MHDL COMPILER

The MHDL compiler converts an MHDL LoC specification and converts it to a netlist, i.e., a graph-based internal data structure in which vertices represent components, and edges represent fluid channels between components. The compiler also collects the library files for the entities used by the LoC.

A. Assembly Language Framework

We developed a mnemonic assembly language on top of the machine language described in Section II.C. The assembly language framework provides a higher-level of abstraction than the machine language for programming the LoC, and abstracts its programming interface away from technology-specific issues. The instructions available in the assembly language framework depend on the functionality of the entities contained in the LoC specification; however, they do not depend on the specific technology. For example, a *mix* instruction can be executed on any LoC that contains a mixer, regardless of the technology in which the LoC is realized.

B. Device-specific Assembler

The *device-specific assembler (DSA)* reads the LoC netlist specification, and generates a graph-based internal data structure to represent the LoC's architecture. The assay is specified using the assembly language framework described in the preceding subsection. The DSA ensures that the LoC is capable of performing each assay operation. The following steps verify the LoC's capability

The assay is input using the assembly language framework. The DSA must then ensure that the device is capable of performing each instruction in the assay. The following steps verify the capability of the device specified to execute all of the instructions in the assay.

First, every component must have the operational capability required by the instruction. If the instruction requires resource *A* to perform a *mix* operation, then the entity type of *A*, as defined in its library file, must support the *mix* operation.

Second, the DSA verifies that all fluid transfers are possible. If the specification includes a transfer from resource *A* to resource *B*, then the DSA must verify that an unobstructed path from *A* to *B* exists in the netlist. An unobstructed path could be a direct connection via a fluid channel, or a component that is not in-use. An obstructed path, in contrast, would go through a component that is in use, and is therefore unable to provide fluid transport capabilities.

IV. DEBUGGING, VERIFICATION, AND SIMULATION

Verifying the correctness of an LoC specification can be difficult, especially as its size and complexity increases. In particular, the issue in question is whether an LoC, specified in MHDL, can correctly execute an assay described using the device assembly language. If so, then the software produces an estimate of the execution time of the assay on the LoC through event-driven simulation; if not, the user has presumably made a mistake, although the tool cannot tell, a-priori, whether the mistake was in the LoC or the assay specification. The user iteratively modifies either specification until functional correctness is achieved, and can make further modifications to tune performance.

A. Simulation

The input to the simulator is the netlist describing the LoC, and the assembly language specification of the assay. The simulator steps through the assay and outputs an execution trace. The simulator accumulates the time take by each assay operation, which is often derived from the assembly language specification (e.g., "Mix A and B for 10 seconds").

The simulator also accounts for fluid transfer time T, which depends on the volumetric flow rate, Q, and the volume of fluid to be transferred, V:

$$T = V/Q. \tag{1}$$

The Hagen-Poisseuile equation [5] can approximate the volumetric flow-rate, Q, as a function of the physical properties of the fluid and the channel:

$$Q = \frac{\Delta P * \pi * d^4}{128 * \mu * L}, \tag{2}$$

where ΔP is the pressure drop, d and L are the diameter and length of the channel, and μ is the fluid viscosity. Parameter values d and L are not known until the LoC is physically laid out, which is beyond the scope of this paper; to enable coarse-grained estimates of fluid transfer latencies, the user can provide estimates for these parameter values for each channel to the simulator.

For debugging purposes, the simulator also outputs an execution trace of the fluids that are moved through the LoC during assay execution. The simulator also outputs a summary describing the fluid volumes remaining in the device at the end of assay execution and a statistical summary of different aspects of the simulation.

B. *Error Checking and Verification*

The simulator presently detects several simple deficiencies that may be present in an LoC. First, it checks the validity of all components used by each assembly instruction. Second, it verifies that all components specified by the instruction can perform the functions required to execute the instruction. Third, it ensures that valid paths in the device exist for all fluid transfer operations. Taken in conjunction with one another, these three conditions determine the *functional correctness* of the LoC. If any of these three tests fails, then either the LoC or assay specification must be revised.

V. Case Study: Influenza Detection LoC

Fig. 4(a) depicts an LoC for influenza detection, which was developed at the University of Michigan [16]. The main components of the LoC are clearly labeled, along with control valves, and connections between the components are readily observed. Fig. 4(b) displays its MHDL specification.

Fig. 5(a) shows an English language description of the influenza assay [16]. We were unable to reproduce the assay with 100% accuracy due to the ambiguity of the original specification, but this inaccuracy does not affect the concepts presented in this paper. Fig. 5(b) shows an assembly language specification of the assay corresponding to the control interface obtained from the MHDL netlist of the LoC shown in Fig. 4(b).

Fig. 6 shows a partial execution trace of the assay running on the LoC, along with a summary of the results. This result shows demonstrates that the MHDL specification of the LoC can execute the assembly language specification of the influenza assay correctly, and that the simulator can estimate the performance of the LoC.

Lastly, we conducted tests where we manually introduced the errors in the MHDL LoC specification, such as removing components and/or connections between components, and attempted to execute the assay. The simulator detected the errors. We also introduced errors into the assay specification, and the simulator detected those errors as well.

VI. Related Work

A. *EDA Support for Microfluidics*

There are a number of competing LoC technologies that are based on actuation technologies other than programmable microvalves. One such technology is electrowetting on dielectric (EWOD), which can manipulate droplets of liquid on a hydrophobic surface lying on top of a two-dimensional array of electrodes [11, 17]. This 2D electrode pattern makes the use of an HDL description of the device unnecessary.

To the best of our knowledge, this is the first work that has presented a microfluidic HDL for microvalve-based LoC's. There has been previous work in the area of automatic routing and control optimization in the form of an AutoCAD plugin, Micado [2]. Micado uses a description of the flow of the chip to automatically infer the location of control valves and to route these valves to external control ports. A tool like Micado could be integrated into a CAD flow that translates our HDL specification into a physical device layout; for now, we will leave this issue open for future work.

B. *LoC Fabrication Technologies*

There have been several groups working on speeding up the fabrication process. One of these groups has been examining separating passive components (fluid channels) from active components (mixers, valves, sensors, etc., which have control) into different helps with fabrication [14]. The process to fabricate the passive layer is simpler, faster, and cheaper than fabricating active layer components. This approach could enable microfluidic analogues to structured ASICs: a generic layer consisting of a variety of active components could be fabricated once and sold to many different users. Each application would then require a different passive layer.

Another approach is to view components as blocks that can be fabricated separately and assembled in the lab for various assays [18, 23]. The blocks will require standard interfaces for fluid transport and control. The advantage is that different experimental apparatuses can be assembled and disassembled using the same set of blocks. Moreover, if a new component is needed for a given assay, it is faster and cheaper to fabricate a new component, rather than an LoC.

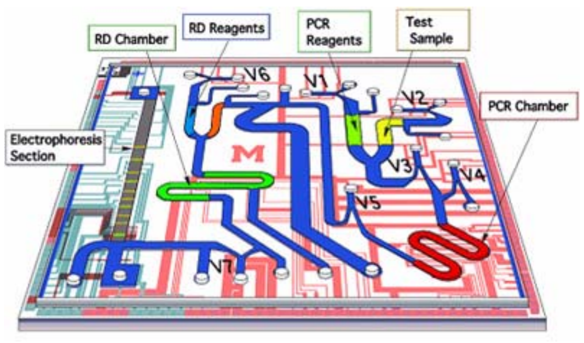
Our MHDL could be used to describe chips using both of these fabrication technologies. The debugging tool would also be able validate and debug configurations before fabrication, or assembly of an apparatus. This would allow scientists to achieve the speedup of both the fabrication process, as well as the design process, without sacrificing accuracy.

C. *Programming Languages and Architectural Support*

EXACT [19] and BioCoder [3] are assay specification languages that attempt to remove ambiguities in English-language specifications. They enforce a strict specification of the assay, and output cookbook-style English languages recipes that are unambiguous.

BioStream [20] allows the user to specify mixing and storage operations, and introduced the notion of volume management. Volume management allows the user to specify fluid quantities in relative, rather than absolute terms, e.g., mix fluids A and B in a ratio of X:Y, but without specifying absolute values for X and Y.

One of our long-term goals is to replace the assembly-level specification of an assay—which requires the programmer to understand the device architecture completely—with a device-independent high-level language specification that is compiled into assembly language. In principle, we could use BioStream, EXACT, or BioCoder as the specification language, and develop a new compiler to achieve our goals.



(a)

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

(b)

Fig. 4. (a) Illustration of an LoC that performs influenza detection [16]; the key observation here is that the LoC can be viewed as a netlist of compnents, which lends itself to an MHDL specification (b).

Influenza Detection Assay

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

(a)

```
load flu_diagnoses
assay flu diagnoses

fill L1,PCR_Reagents,240,.0008
fill L2,DNA,600,.0008
fill L3,RDReagents,500,.0008
fill B4,ReproGel,100,.0008

moveper PCR_chamber,L1,100
moveper PCR_chamber,L2,100

heat PCR_chamber,30,92

repeat loop 35 times
   heat PCR_chamber,5000,92
   heat PCR_chamber,10000,55
   heat_PCR_chamber,20000,72
end loop

heat PCR_chamber,60000,72

moveper PCR_product, PCR_chamber,50
moveper RD_chamber,L3,100
moveper RD_chamber,PCR_product,100

digest RD_chamber,600000,37
moveper B1,RD_chamber,50

moveper electrophoresis_section,B1,100
moveper electrophoresis_section,B4,100
separate electrophoresis_section,600000

moveper B3,electrophoresis_section,100
flush B3
```

(b)

Fig. 5. (a) English language and (b) assembly language specification of the influenza detection assay.

AquaCore [1] is a programmable LoC architecture with a fluidic instruction set architecture. Aquacore's compiler automatically translates high-level assay specifications into an executable format. Our idea to derive a machine-language interface to an MHDL specification was inspired by AquaCore. In principle, we could specify a device like AquaCore using our MHDL, although thus far, we have focused on specifying assay-specific devices instead.

```
/* Execution Trace */
L1 is filled with 240 cubic microns of PCR Reagents
L2 is filled with 600 cubic microns of DNA
L3 is filled with 500 cubic microns of RDReagents
B4 is filled with 100 cubic microns of ReproGel

Moved 240 cubic microns from L1 to PCR_chamber
   L1->L1V1(65.2229)->V1(0)->V1V3(326.115)->V3(0)->
      V3V4(130.446)->V4(0)->PCR_chamber
   L1 is now empty
   PCR_chamber now contains 240 cubic microns of
      PCR_Reagents
Time: 891 ms
…

/* Remaining Fluids */
PCR_chamber contains 300 cubic microns of DNA
RD_chamber contains 150 cubic microns of DNA

/* Statistics Tracking */
Total Valve Actuation Time:        2400ms
Total Time in Lines:               2380.64ms
Total Transfer Time:               5224ms
Total Time in Components:          2.48507e+06ms
Total Time:                        2.49049e+06ms
Total Number of Valve Actuations:  17
```

Fig. 6. Partial execution trace and summary output of the simulator executing the assembly language specification of the influenza detection assay (Fig. 5(b)), on the MHDL specification of the influenza detection LoC (Fig. 4(b)).

VII. Conclusion and Future Work

This paper has introduced the MHDL language for LoC specification, and the software toolchain that enables its usage. Rudimentary mechanisms for assembly language assay specification, compilation, simulation, and verification and debugging were introduced and shown to be effective. The authors strongly believe that this approach represents the most effective way to transition to HDL-based design of continuous fluid-flow-based LoCs. Our future work will add automatic physical layout capabilities to our tools, which will provide a direct path from MHDL specification to LoC fabrication.

Acknowledgments

This work was supported by United States National Science Foundation Grant CNS-1035603. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

References

[1] A. M. Amin et al. AquaCore: A programmable architecture for microfluidics. Int. Symposium on Computer Architecture (ISCA), 2007, pp. 254-265.

[2] N. Amin et al. Computer-aided design for microfluidic chips based on multilayer soft lithography. IEEE International Conference on Computer Design (ICCD) 2009, pp. 2-9.

[3] V. Ananthanarayanan and W. Thies. Biocoder: A programming language for standardizing and automating biology protocols. Journal of Biological Engineering, 4:13 (2010).

[4] I. E. Araci and S. R. Quake. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. Lab on a Chip 12:2803-2806 (2012)

[5] H. Bruus. Theoretical Microfluidics. Oxford Univ. Press, USA

[6] N. S. G. K. Devaraju and M. A. Unger: Pressure driven digital logic in PDMS based microfluidic devices fabricated by multilayer soft lithography," Lab on a Chip 12:4809-4815 (2012)

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

[8] D. Gajski. NISC: the ultimate reconfigurable component. Technical report TR 03-28, Center for Embedded Computer Systems, UC Irvine, October 2003.

[9] W. H. Grover et al. Developed and multiplexed control of latching pneumatic valves using microfluidic logical structures. Lab on a Chip 6:623-631 (2006)

[10] W. H. Grover et al. Monolithic membrane valves and diaphragm pumps for practical large-scale integration into glass microfluidic devices. Sensors and Actuators B., 89:315-323 (2003)

[11] T. Ho, et al. Digital microfluidic biochips: recent research and emerging challenges. Conference on Codesign and System Synthesis (CODES-ISSS), 2011, pp. 335-343.

[12] E. C. Jensen et al. A digital microfluidic platform for the automation of quantitative biomolecular assays. Lab on a Chip 10:685-691 (2010)

[13] K. Keutzer, A. R. Newton, J. M. Rabaey, and A. L. Sangiovanni-Vincentelli: System-level design: orthogonalization of concerns and platform-based design. IEEE Trans. on CAD of Integrated Circuits and Systems 19(12): 1523-1543 (2000)

[14] J. Melin and S. R. Quake. Microfluidic large-scale integration: the evolution of design rules for biological automation. Annu. Rev. Biophys. Biomol. Struc., 36:213-231 (2007).

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

[16] R. Pal et al. An integrated microfluidic device for influenza and other genetic analyses. Lab on a Chip, 5:1024-1032 (2005)

[17] M. G. Pollack, et al. Electrowetting-based actuation of droplets for integrated microfluidics. Lab on a Chip, 2:96-101 (2002)

[18] M. Rhee and Mark A. Burns. *Microfluidic assembly blocks.* Lab on a Chip, 8:1365-1373 (2008)

[19] L. N. Soldatova et al. The EXACT description of biomedical protocols. BioInformatics, 24: i296 – i303 (2008)

[20] W. Thies et al. Abstraction layers for scalable microfluidic Biocomputing. Natural Computing, May 2007.

[21] J. P. Urbanski et al. Digital microfluidics using soft lithography. Lab on a Chip, 6:96-104 (2006)

[22] Y. Xia, and G. M. Whitesides. Soft lithography. Angew. Chem. Int. Ed. Engl., 37(5):551-575 (1998)

[23] P. K. Yuen. SmartBuild—a truly plug-n-play modular microfluidic system. Lab on a Chip 8:1374-1378 (2008)