Automatic Synthesis of Microfluidic Large Scale Integration Chips from a Domain-Specific Language

Jeffrey McDaniel, Christopher Curtis, Philip Brisk
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA, 92521

***Abstract*—BioCoder is a domain-specific language by which chemists and biologists can express experimental protocols in a manner that is unambiguous and clearly repeatable. This paper presents a software toolchain that converts a protocol specified in a restricted subset of BioCoder to a technology-specific description of the protocol, targeting flow-based microfluidic large-scale integration (mLSI) chips. The technology-specific description can then be used to either: (1) execute the protocol on a capable chip; or (2) to derive the architecture of a new mLSI chip that can execute the protocol.**

Keywords—microfluidic Large Scale Integration (mLSI), domain-specific language

I. INTRODUCTION

Emerging *laboratory-on-a-chip* technology provides precise control of micro- to nano-liter quantities of fluids, which can be used to automate a wide variety of biochemical assays. In recent years, microfabrication techniques have matured, enabling the fabrication of smaller and increasingly complex devices. We refer to such devices here as *microfluidic large scale integration (mLSI)* chips [4, 7], due to a rapid increase in the number of externally controlled microvalves that can be integrated into a single chip, which parallels *Moore's Law* in the semiconductor industry.

At present, designers specify mLSI chips by hand using AutoCAD. This task is tedious and may take weeks or months, depending on the complexity of the chip. Manual design is non-scalable and is quickly becoming impractical for all but the simplest mLSI chips. Moreover, the potential user base for mLSI chips are chemists and biologies, who understand the assays that they wish to execute, but are not device architects themselves. As such, there is a clear and unequivocal need for synthesis tools that can automatically generate and lay out an mLSI chip architecture from a high-level device-agnostic specification of an assay.

This paper introduces an automatic synthesis tool that provides this capability. The user of the tool specifies the assay using a restricted subset of a domain-specific language for biological protocols called BioCoder [3]. The synthesis tool compiles the BioCoder assay specification and produces: (1) a description of an mLSI chip architecture that can execute the assay; and (2) a sequence of microvalve actuations that can execute the protocol on the chip.

A. Microfluidic Design Automation (MDA)

The semiconductor industry introduced *electronic design automation (EDA)* in the early 1980s, which has enabled the production of complex integrated circuit designs whose complexity wholly eclipses what is possible with manual design [6]. The objective of this work is to enable *microfluidic design automation (MDA)*, which will provide a similar productivity boon to the designers of mLSI chips.

Several academic research groups have published papers describing MDA algorithms, but an end-to-end toolflow, from specification to physical layout, has not yet been demonstrated. These tools require the assay to be specified as a *sequencing graph* [8]. In our view, a sequencing graph is an appropriate internal representation for an automatic synthesis tool to employ, but is an unwieldy and poorly-defined language for assay specification, especially when the number of operations and dependencies become too large for a person to draw on a single page.

This paper describes a fully automated synthesis flow targeting mLSI technology in which assays are specified using a domain-specific language in lieu of a sequencing graph; the focus is the high-level compilation steps and intermediate languages used to represent the protocol.

B. Toolflow

Figure 1 illustrates the automatic synthesis flow. It makes use of a lower-level compiler and simulation environment [5], as well as a collection of algorithms that synthesize and physically layout the mLSI chip, starting from a sequencing graph specification [2, 8]. The assay is specified using an mLSI-compatible subset of the BioCoder language [3] (*Section II.A*). The BioCoder compiler outputs a sequencing graph, which is input to the physical layout flow, as well as an equivalent specification of the protocol in a mid-level *virtualized assembly language* (*Section II.B*), which is then scheduled onto the mLSI chip architecture, producing a low-level *device-specific assembly language* (*Section II.C*) specification. The compiler and simulation environment [5] then verifies the correctness of the mLSI chip, estimates its performance, and outputs a program to control the device.

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

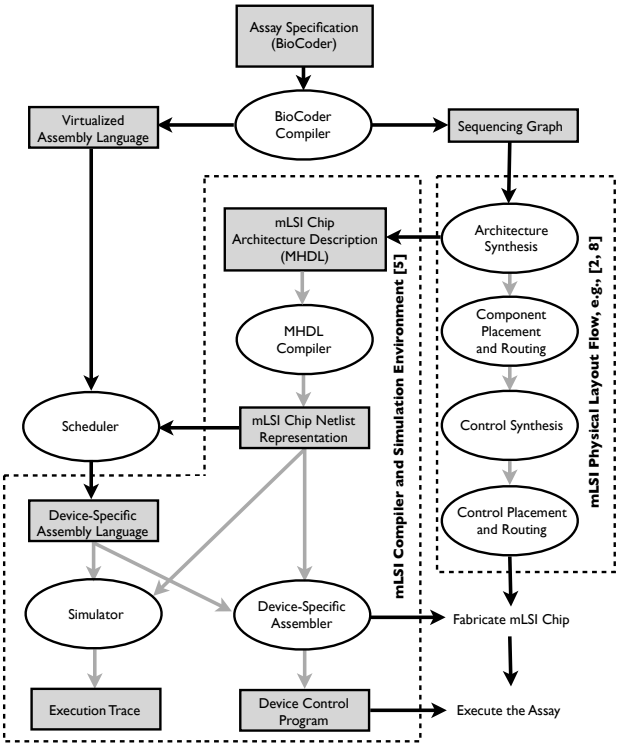

Fig. 1. Oveview of the software toolflow outlined in this paper. The framework makes use of a lower-level compiler and simulation environment [5] and physical layout flow [2, 8]. Gray boxes represent text files (several of which are human-readable) that are passed between the software modules.

II. Running Example

We present a running example, an influenza detection assay [9], which illustrates the key contributions of this paper. Figure 2 presents an English-language specification of the assay. Figure 3 presents its BioCoder specification, and Figures 4 and 5 present virtualized and domain-specific assembly language specifications, which are generated automatically from our software toolflow; Ref. [5, Fig. 4(b)] presents an MHDL description of an mLSI chip capable of executing the assay (used by the Scheduler in Figure 1).

A. BioCoder

BioCoder is a C library developed at Microsoft Research, India, which is intended to standardize the specification and dissemination of biological protocols [3]. A biologist specifies the protocol in C using the BioCoder library; the BioCoder compiler then outputs a step-by-step specification of the protocol in a manner that is fully unambiguous; the output has the conceptual look and feel of a recipe in a cookbook. This unambiguous recipe can then be disseminated to other biologists in order to enhance clarity and reproducibility within the larger research community.

The paper that introduced BioCoder briefly mentioned the potential development of a compiler targeting a microfluidic lab-on-a-chip, but left that aspect of the research open for future work; this paper describes one such implementation.

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.

Fig. 2. English language specification of an influenza detection assay [9].

This work makes several modifications to the BioCoder language. First, we extended the syntax so that it has the flavor of C++, rather than C. Secondly, we eliminated support for aspects of the language that are not compatible with lab-on-a-chip technology, for example, solid substances. At present, BioCoder does not support control flow operations, so all loops are explicitly unrolled.

The BioCoder specification of an assay makes no underlying assumption about the target technology (mLSI chips, or something else) or about the target architecture (e.g., how many mixers are available, interconnection schemes, etc.). A device-specific compiler that targets known mLSI chips must determine if that chip has sufficient architectural components and interconnections to execute the protocol [5]. In contrast, an automatic synthesis tool can take a BioCoder specification of an assay and design an application-specific mLSI chip that can execute the protocol.

Figure 3 presents the BioCoder specification of the influenza detection assay. The fluids are first initialized with their respective names and volumes. Physical resources that perform operations are viewed as "abstract containers," to designate storage of fluid; the synthesis process will allocate physical containers in the mLSI chip which will store the physical fluid. The remainder of the BioCoder program specifies the actions to be performed; these actions can readily be converted to a sequencing graph to represent fluid dependencies between operations. These dependencies impose a partial ordering on the operations. As the mLSI chip architecture is determined algorithmically by the toolflow, a schedule will eventually be computed that obeys both this partial order and the resource constraints of the mLSI chip that will execute the assay.

B. BioCoder Compiler: Virtualized Assembly Language and Sequencing Graph Assay Specifications

The BioCoder compiler accepts a BioCoder specification of the protocol as input and converts it to a virtualized assembly language specification used for further stage of compilation, and a semantically equivalent sequencing graph representation used by the physical synthesis flow, as shown in Figures 4 and 5, respectively.

The virtualized assembly language is human readable, and is device agonistic, like BioCoder; compared to BioCoder, it is a lower-level representation and far less verbose. Figure 4 examplifies the correspondence between the BioCoder and virtualized assembly language specification in our example.

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

        FluDiagnosis->measure_fluid(F1,Volume(240,UL,tube1);
        FluDiagnosis->measure_fluid(F2,Volume(600,UL,tube1);
        FluDiagnosis->vortex(tube1,Time(1,SECS);
        FluDiagnosis->incubate(tube1, 92, Time(30, SECS));

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

        FluDiagnosis->measure_fluid(F3, (Volume(500,UL), tube1));
        FluDiagnosis->vortex(tube1, Time(1, SECS));
        FluDiagnosis->store_for(tube1, 37, Time(600000, SECS));

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

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

Fig. 3. BioCoder specification of the influenza detection assay.

```
assay FluDiagnosis
mix MIX1, PCR_REAGENTS, 240, DNA, 600, 1
heat HT_1, MIX1, 840, 30, 92
repeat loop 35 times
     heat HT_2, HT_1, 840, 5000, 92
     heat HT_3, HT_2, 840, 10000, 55
     heat HT_4, HT_3, 840, 20000, 72
end loop
heat HT_107, HT4, 840, 60000, 72
mix MIX2, HT_107, 840, RD_Reagents, 500, 1
heat HT_108, MIX2, 1340, 600000, 37
mix MIX3, HT_108, 1340, ReproGel, 100, 1
electrophoresis DCT1, MIX3, 1440, 5
```

Fig. 4. Virtualized assembly language specification of the influenza detection assay. Compared to the BioCoder specification, the virtualized assembly language specification is lower-level and far less verbose, while still being human-readable.

The virtualized assembly language specification of the assay is input to the scheduler (described in the next section); however, the scheduler cannot proceed until it obtains a description of the mLSI chip architecture (e.g., the number of components that are available, how they are connected, etc.). This information is obtained through the process of architectural synthesis [8], which allocates components and instantiates a fluidic interconnection network between them. The architectural synthesis step of the physical design flow outputs a human-readable specification of the mLSI chip using the MHDL language [5, Fig. 4(b)].

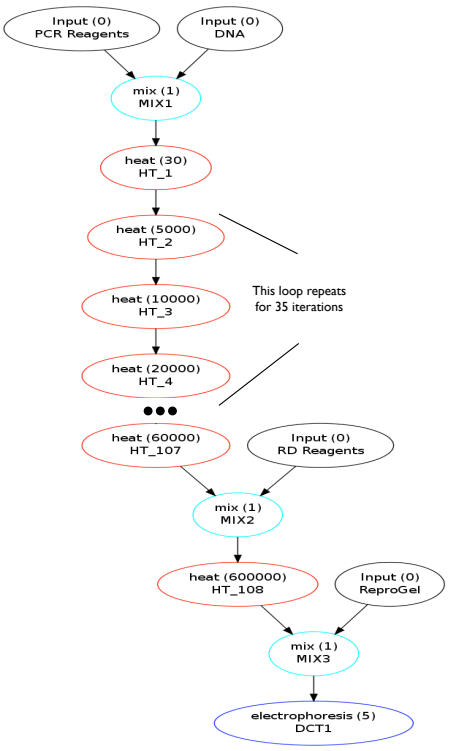

Fig. 5. Sequencing graph specification of the influenza detection assay (equivalent to the virtualized assembly language specification).

```
assay flu_diagnosis
fill Input0, PCR_Reagents, 240
fill Input1, DNA, 600
fill Input2, RD_Reagents, 500
fill Input3, ReproGel, 100
moveper PCR_mixer, Input0, 100
moveper PCR_mixer, Input1, 100
mix PCR_mixer, 1
moveper PCR_heater, PCR_mixer, 100
heat PCR_heater, 30, 92
repeat loop 35 times
     heat PCR_heater, 5000, 92
     heat PCR_heater, 10000, 55
     heat PCR_heater 20000, 72
end loop
heat PCR_heater 60000, 72
moveper RD_chamber, Input2, 100
moveper RD-chamber, PCR_heater, 50
digest RD-chamber, 60000, 37
moveper B1, RD_chamber, 50
moveper electrophoresis_section, RD_chamber, 50
moveper electrophoresis_section, Input3, 100
separate electrophoresis_section, 600000
```

Fig. 6. Device-specific assembly language specification of the influenza detection assay

The MHDL specification of the mLSI chip is then converted into a *netlist* representation, which is an internal data structure used by the compiler. In traditional EDA, the term *netlist* represents a circuit as a set of connections between logic gates (and, hierarchically, larger components that are themselves netlists). In the context of MDA, the netlist represents fluidic components and the fluidic interconnects between the components. Converting an MHDL specification of an mLSI chip into a netlist is straightforward and deterministic [5].

C. Scheduler and Device-Specific Assembly Language

The scheduler compiles a virtualized assembly language specification of an assay onto an mLSI chip specified by a netlist. The scheduler must satisfy both fluidic dependency and resource constraints. In our toolflow, the mLSI chip is derived from the BioCoder specification, as described in the previous two subsections, while making use of the architectural synthesis stage of a physical design flow [8]. In principle, it is also possible to schedule an assay onto an arbitrary mLSI chip chosen by the user, as long as a netlist specification is available.

In the former case, a legal schedule is guaranteed to exist because the architecture is derived from the assay specification itself. In the latter case, the mLSI chip may or may not be capable of executing the assay. For example, the mLSI chip may not have one (or more) of the necessary components capable of performing some assay steps, or, alternatively, one or more required fluid transfers may not be possible due to the fluidic interconnection scheme.

Our scheduler uses list scheduling [8], an efficient and greedy heuristic that produces fairly good quality schedules quickly; in principle, other scheduling algorithms could be developed, although doing so is beyond the scope of this paper. The output of the scheduler is a device-specific assembly language specification of the assay, as shown in Figure 6. This specification includes detailed information about which physical resources on the mLSI chip execute each fluidic operation, and fluid transfers between physical resources. Techniques to determine routing paths for fluids and to generate a sequence of microvalve actuations from a device-specific assembly language specification to execute the protocol on an mLSI chip are known [5].

III. Related Work

There have been several attempts in recent years to create programming languages for biochemistry, some more successful than others. This research effort builds on the BioCoder language [3], which itself is an evolution of a prior language called *BioStream* [11]. Biostream was targeted toward a domain-specific mLSI chip architecture and instruction set designed to perform serial dilutions [12].

AquaCore [1] is a general-purpose mLSI chip architecture that features a collection of components that are connected by a fluidic bus; the *AquaCore Instruction Set (AIS)* is similar to the device-specific assembly language presented here, but remains tied to the AquaCore architecture. Unlike BioStream and AIS, the device-specific assembly language introduced here is not coupled to a specific mLSI chip architecture, and therefore offers greater flexibility and generality.

EXACT [10] is a language designed to standardize dissemination of biological protocols. It uses an HTML-style tagging language for assay specification. In our opinion, the assays quickly become long and overly verbose, making them difficult to read. For this reason, we chose to work with BioCoder instead.

MHDL, in contrast, is a specification language for mLSI chip architectures, as opposed to assay specification [5]. As shown in Figure 1, our toolflow is integrated with a larger compiler and simulation environment for mLSI chips that takes MHDL specifications as an input. Taken in conjunction with existing mLSI physical deisgn flows [2, 8], this paper describes *the first* path from a high-level specification of an assay (using BioCoder) to a physical layout of an mLSI chip, in preparation for fabrication.

IV. Conclusion

Ideally, chemists and biologists should focus on science, rather than the design, verification, and test of complex instruments that automate the process of data collection and analysis. The software presented in this paper will help mLSI chip designers cope with growing device complexity: a scientist can specify an assay using a high-level language, and quickly obtain the schematic diagram for a fully functional mLSI chip that can execute the assay as specified. This toolflow offers a productivity boost to scientists who design mLSI chips to further their research, and lowers the barrier to entry for scientists who wish to employ mLSI chips, but lack the time or expertise required to design the chips on their own.

Acknowledgment

This material is based upon work supported by the U.S. National Science Foundation (NSF) under grant 1035603. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

[1] A. M. Amin, et al., “AquaCore: a programmable architecture for microfluidics,” ISCA, 2007, pp. 254-265

[2] N. Amin, W. Thies, and S. P. Amarasinghe, “Computer-aided design for microfluidic chips based on multilayer soft lithography,” 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] J. McDaniel, et al., “Design and verification tools for continuous fluid flow-based microfluidic devices,” ASPDAC 2013, pp. 219-224.

[6] C. Mead and L. Conway, Introduction to VLSI Systems. Boston, MA: Addison-Wesley, 1979.

[7] 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).

[8] W. H. Minhass, System-Level Modeling and Synthesis Techniques for Flow-Based Microfluidic Very Large Scale Integration Biochips,” Ph.D. Thesis, Technical University of Denmark, 2012.

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

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

[11] W. Thies, et al., “Abstraction layers for scalable microfluidic biocomputing,” Natural Computing, 7(2):255-275 (2008)

[12] J. P. Urbanski, et al., “Digital microfluidics using soft lithography,” Lab-on-a-Chip 6(1): 96-104 (2006)