Architectural Synthesis of Flow-Based Microfluidic Large-Scale Integration Biochips

Wajid Hassan Minhass
whmi@imm.dtu.dk

Paul Pop
pop@imm.dtu.dk

Jan Madsen
jan@imm.dtu.dk

Felician Stefan Blaga

DTU Informatics
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark

ABSTRACT

Microfluidic biochips are replacing the conventional biochemical analyzers and are able to integrate the necessary functions for biochemical analysis on-chip. In this paper we are interested in flow-based biochips, in which the flow of liquid is manipulated using integrated microvalves. By combining several microvalves, more complex units, such as micropumps, switches, mixers, and multiplexers, can be built. The manufacturing technology, soft lithography, used for the flow-based biochips is advancing faster than Moore's law, resulting in increased architectural complexity. However, the designers are still using full-custom and bottom-up, manual techniques in order to design and implement these chips. As the chips become larger and the applications become more complex, the manual methodologies will not scale, becoming highly inadequate. Therefore, for the first time to our knowledge, we propose a top-down architectural synthesis methodology for the flow-based biochips. Starting from a given biochemical application and a microfluidic component library, we are interested in synthesizing a biochip architecture, i.e., performing component allocation from the library based on the biochemical application, generating the biochip schematic (netlist) and then performing physical synthesis (deciding the placement of the microfluidic components on the chip and performing routing of the microfluidic channels), such that the application completion time is minimized. We evaluate our proposed approach by synthesizing architectures for real-life applications as well as synthetic benchmarks.

Categories and Subject Descriptors

J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms

Design, Performance

Keywords

microfluidic, biochips, architecture, synthesis, flow-based

1. INTRODUCTION

Microfluidics-based biochips (also referred to as lab-on-a-chip) integrate different biochemical analysis functionalities (e.g., mixers, filters, detectors) on-chip, miniaturizing the macroscopic biochemical processes to a sub-millimetre scale [12]. These microsystems offer several advantages over the conventional biochemical analyzers, e.g., reduced sample and reagent volumes, faster biochemical reactions, ultra-sensitive detection and higher system throughput, with several assays being integrated on the same chip [15].

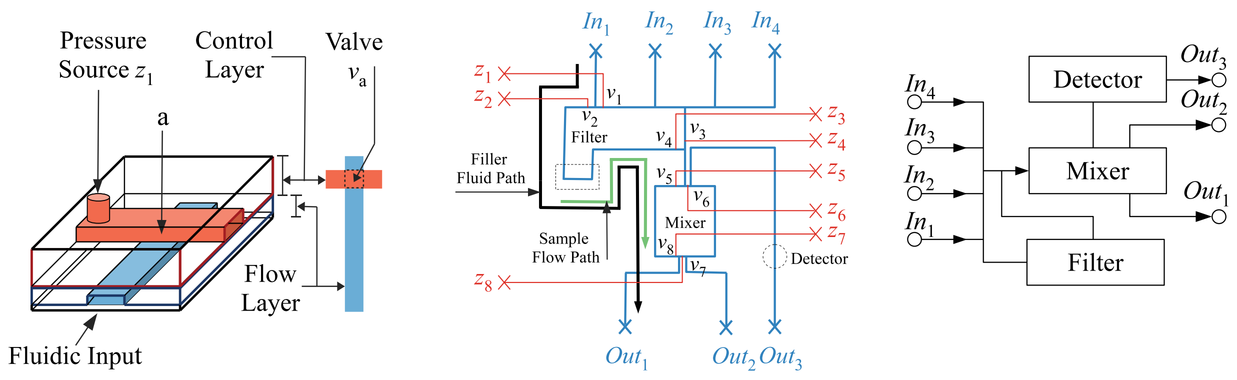
Microfluidics-based biochips have become an actively researched area in recent years. These chips can readily facilitate clinical diagnostics, especially immediate point-of-care disease diagnosis. In addition, they also offer exciting application opportunities in the realm of massively parallel DNA analysis, enzymatic and proteomic analysis, cancer and stem cell research, and automated drug discovery [12, 15]. Utilizing these biochips to perform food control testing, environmental (e.g., air and water samples) monitoring and biological weapons detection are also interesting possibilities.

There are several types of biochip platforms, each having its own advantages and limitations [11]. In this paper, we focus on the flow-based biochips in which the microfluidic channel circuitry on the chip is equipped with chip-integrated microvalves that are used to manipulate the on-chip fluid flow [12]. By combining several microvalves, more complex units such as mixers, micropumps, multiplexers can be built, with hundreds of units being accommodated on a single chip [11]. Analogous to its microelectronics counterpart, this approach is called microfluidic Large Scale Integration (mLSI) [12].

1.1 Related Work

During the last decade, a significant amount of work has been carried out on the individual microfluidic components as well as the microfluidic platforms [10, 11]. The manufacturing technology, soft lithography, used for the flow-based biochips has advanced faster than Moore's law [9]. Although biochips are becoming more complex everyday, Computer-Aided Design (CAD) tools for these chips are still in their infancy. Most CAD research has been focussed on device-level physical modeling of components [17].

Designers are using full-custom and bottom-up methodologies to implement these chips. Microfluidic components are designed and connected together to match the steps of the desired biochemical application using technical drawing tools such as AutoCAD [1]. In order to design a chip, the designer needs to have a complete understanding of the application requirements and at the same time, have the knowledge and skills of chip fabrication. The placement and routing is also done manually [2] and then the chip is fabricated using soft lithography techniques. Recent work has proposed automation techniques for the placement, routing and optimization



(a) Microfluidic Valve (b) Biochip: Schematic View (c) Biochip: Functional View

Figure 1: Flow-Based Biochip Architecture Model

for the channels that are used to control the microvalves [4]. However, this work is limited to the control part of the chip. In order to execute the application, designers manually map the operations to the valves of the chip using a customized interface (analogous to exposing the gate-level details).

As the chips grow more complex (commercial biochips are available which use more than 25,000 valves and about a million features to run 9,216 polymerase chain reactions in parallel [15]) and the need of having multiple and concurrent assays on the chip becomes more significant, the manual methodologies will not scale and will become highly inadequate. Therefore, new top-down methodologies and design tools are needed, in order to provide the same level of CAD support to the biochip designer as the one currently taken for granted in the semiconductor industry. Such an approach would also decouple the biochip architecture design from fabrication details, allowing users to focus on applications without requiring knowledge and skills of chip fabrication [12].

Significant work on top-down synthesis methodologies for droplet-based biochips has been proposed [6]. However, the architecture of the droplet-based chips differs significantly from the flow-based chips. In the flow-based biochips, components of different types (e.g., mixers, heaters) are physically designed on the chip and connected to each other using microfluidic channels. Once fabricated, the number and type of the components, their placement scheme on the chip and the routing interconnections cannot be modified. Droplet-based biochips (also referred to as digital biochips), however, use the idea of virtual components and are reconfigurable. A digital biochip consists of a two-dimensional array of identical electrodes on which the fluid is manipulated as discrete droplets. Adjacent set of electrodes can be combined together to form a virtual component, e.g., a mixer can be created by grouping adjacent electrodes and moving the droplet around on these electrodes to achieve mixing. Any set of electrodes can be used for this purpose and thus the chip is termed reconfigurable. The same electrodes can later be used for performing other operations as well, e.g., fluid transport, storage. Because of the architectural differences, the models and techniques proposed for the digital chips are not applicable to their flow-based counterparts.

1.2 Contribution

We propose a top-down architectural synthesis methodology for the flow-based microfluidic biochips. Given a biochemical application modeled as a sequencing graph, a microfluidic component library and the chip area, the architectural synthesis consists of the following three steps: (i) *allocation* of components from a given library, (ii) performing the *schematic design* and generating the *netlist*, and the biochip (iii) *physical synthesis*, i.e., deciding the placement of the microfluidic components on the chip and performing routing of the microfluidic channels on the available routing layers creating component interconnections.

The synthesis problem is NP-complete. We use an approach similar to High-Level Synthesis [8] for performing allocation and netlist generation. The component placement is done using Simulated Annealing and we tailor the Hadlock's algorithm [16] from the Very Large-Scale Integrated (VLSI) circuits domain for performing the microfluidic channel routing. Synthesis is done in such a way that the application completion time is minimized and the imposed constraints (e.g., resource, dependency) are satisfied.

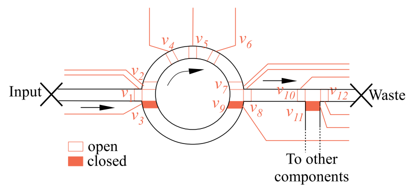
We build on our previous work in [14], where we consider that the architecture is given, and propose an approach for mapping a biochemical application onto the given architecture such that the application completion time is minimized. However, this paper is the first to present an approach for the automatic synthesis of a biochip architecture. The main contributions of this paper are the formulation of the architectural synthesis problem and the proposed synthesis framework, which show how the well-known algorithms from the High-Level Synthesis of VLSI circuits can be tailored to tackle the flow-based biochips.

The paper is organized in seven sections. The biochip architecture model and the biochemical application model are presented in Section 2. The problem is formulated in Section 3 and the synthesis steps are presented in Section 4. The proposed synthesis framework is discussed in Section 5 and is further evaluated in Section 6. We present our conclusions in Section 7.

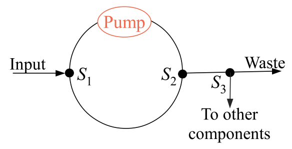
2. SYSTEM MODEL

2.1 Biochip Architecture

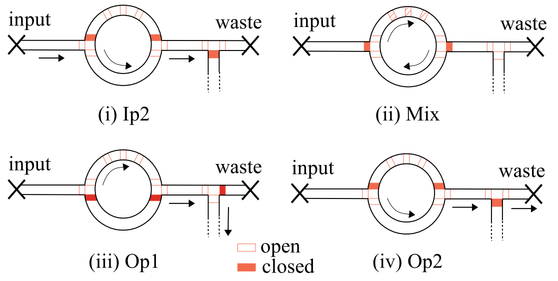
Fig. 1b shows the schematic view of a flow-based biochip with 4 input ports and 3 output ports, 1 mixer, 1 filter and 1 detector. Fig. 1c shows the functional view of the same chip. The biochip is manufactured using multilayer soft lithography [12]. A cheap, rubber-like elastomer (polydimethylsiloxane, PDMS) with good biocompatibility and optical transparency is used as the fabrication substrate. Physically, the biochip can have multiple layers, but the layers are logically divided into two types: *flow layer* (depicted in blue) and the *control layer* (depicted in red). The liquid in the flow layer is manipulated using the control layer [12].


Input
Waste
v_1 v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} v_{12}
open
closed
To other components

(a) Schematic View, Ip1



(b) Conceptual View



(c) Operational Phases

Figure 2: Microfluidic Mixer

The basic building block of such a biochip is a valve (see Fig. 1a), which is used to manipulate the fluid in the flow layer as the valves restrict/ permit the fluid flow. The control layer (red) is connected to an external air pressure source z_1. The flow layer (blue) is connected to a fluid reservoir through a pump that generates the fluid flow. When the pressure source is not active, the fluid is permitted to flow freely (open valve). When the pressure source is activated, high pressure causes the elastic control layer to pinch the underlying flow layer (point *a* in Fig. 1a) blocking the fluid flow (closed valve). Because of their small size (100×100 μm^2), a biochip can accommodate thousands of valves. By combining these valves, more complex units, such as switches, multiplexers, micropumps, mixers, can be built [12]. For example, the valves can be combined to represent a switch. As shown in Fig. 3, a switch may consist of one valve (restricting/ allowing flow in a channel) or may consist of more than one valve. Multiple valve switches are present at the channel junctions and are used to control the path of the fluids entering the switch from different sides. The fluid flow can be generated using off-chip or on-chip pumps. The control layer can be placed both above and/ or below the flow layer, creating "push-down" or "push-up" valves, respectively. Connections to the external ports (fluidic ports and pressure sources) are made by punching holes in the chip (gaining access to the flow and control layer) and placing external tubings (connected to the external fluidic reservoirs through pumps or pressure sources) into the punch holes [12]. All input ports are connected to off-chip pumps.

All fluid samples inside the chip occupy a fixed unit length (or a multiple of it) on the flow channel, i.e., the fluid samples have discretized volumes. Unit length samples are obtained by a process called *metering*, carried out by transporting the sample between two valves that are a fixed length apart [19]. In general, the chip is filled with a filler fluid (e.g., immiscible oil) and the fluid samples are emulsified in the filler fluid. As emulsions, the samples do not touch the channel walls directly (preventing cross-contamination) and can be moved over long channel lengths of any shape while retaining their content [19].

In order to make a fluid sample flow on the chip (e.g., *Filter* to the *Mixer* in Fig. 1b), (i) the point of fluid sample origin (*Filter*) needs to be connected to a pump (on-chip or off-chip) for generating the flow. As shown in Fig. 1b, the closest pump from the *Filter* is the off-chip pump connected to the input port In_1. We term the flow starting point as the *source* (In_1 in this case). (ii) The fluid sample destination point (*Mixer*) needs to be connected to a fluidic output port (*sink*, e.g., Out_1). Next, (iii) a path for the fluid flow needs to be established from the source to the sink using the microfluidic valves and then (iv) the desired flow (*Filter* to *Mixer*) can be achieved by activating the pump. For the *Filter* to *Mixer* flow in Fig. 1b, the path is established by closing the valve

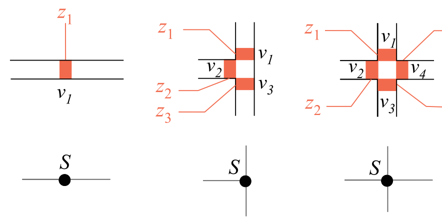


Figure 3: Switch Configurations

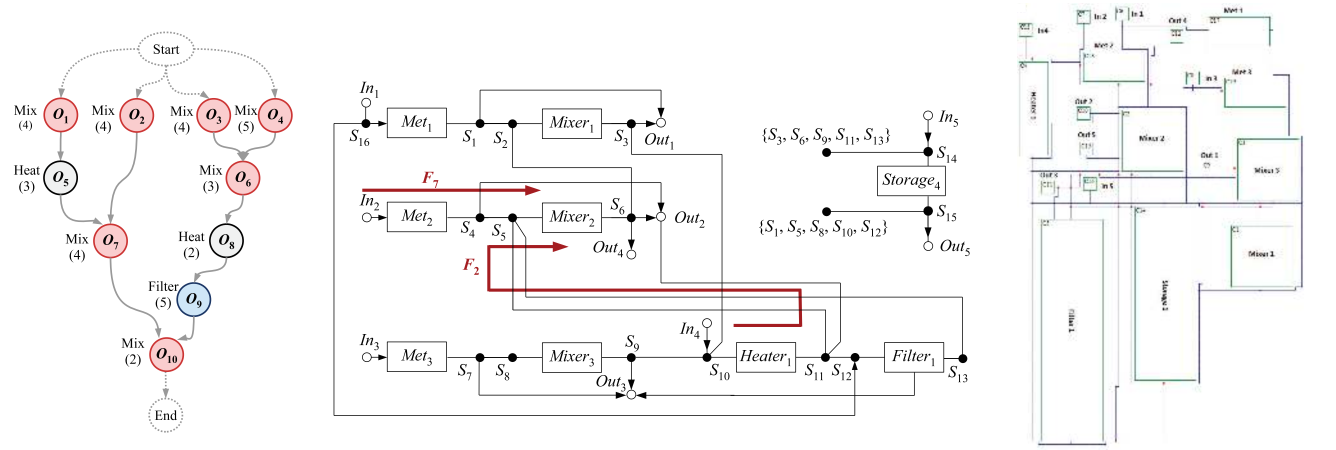
set v_1, v_3, v_6 and v_7, while the valve set v_2, v_4, v_5 and v_8 is kept open (the path is shown in black in Fig. 1b). The entire path already contains the filler fluid and the sample emulsified in the filler fluid is now present inside the *Filter*. A pumping action at the source (In_1) then creates a filler fluid flow towards the sink (Out_1). The emulsified sample flows with the filler fluid from the *Filter* towards the *Mixer*. The pumping action is stopped once the fluid sample reaches its destination (the green path in Fig. 1b shows the flow of the sample). While the sample flows from the *Filter* to the *Mixer*, the established path (including the source, sink points) is reserved and cannot be used for any other flows.

2.1.1 Component Model

We use our dual-layer component modeling framework proposed in [14] consisting of a *flow layer model* and a *control layer model*. The flow layer model ($\mathcal{P}, C, \mathcal{H}$) of each component M is characterized by a set of operational phases $\mathcal{P}$, execution time C and the component geometrical dimensions $\mathcal{H}$. The control layer model captures the valve actuation details required for the on-chip execution of all operational phases of a component. Table 1 shows the flow layer model library $\mathcal{L} = M(\mathcal{P}, C, \mathcal{H})$ of eight commonly utilized microfluidic components [10, 19].

Table 1: Component Library ($\mathcal{L}$): Flow Layer Model

Component	**Phases (P)**	**Exec. Time (C)**
Mixer	Ip1/ Ip2/ **Mix**/ Op1/ Op2	0.5 s
Filter	Ip/ **Filter**/ Op1/ Op2	20 s
Detector	Ip/ **Detect**/ op	5 s
Separator	Ip1/ Ip2/ **Separate**/ Op1/ Op2	140 s
Heater	Ip/ **Heat**/ Op	20°C/s
Metering	Ip/ **Met**/ Op1/ Op2	-
Multiplexer	Ip or Op	-
Storage	Ip or Op	-



(a) Application Graph (b) Biochip Architecture (c) Placement and Routing

Figure 4: Biochip Application and Architecture Example

Table 2: Mixer: Control Layer Model

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

Consider the pneumatic mixer [7] in Fig. 2a which is implemented using nine microfluidic valves, v_1 to v_9. Fig. 2b shows the conceptual view of the same mixer. The valve set $\{v_4, v_5, v_6\}$ acts as an on-chip pump. The valve set $\{v_1, v_2, v_3\}$ is termed as switch S_1 and the valve set $\{v_7, v_8, v_9\}$ as switch S_2. The two switches facilitate the inputs and outputs, and the pump is used to perform the mixing. The mixer output can either be sent to the waste or to the other components in the chip using the switch S_3 (Fig. 2a).

The mixer has five operational phases. The first two phases represent the input of two fluid samples that need to be mixed, followed by the mixing phase. The mixed sample is then transported out of the mixer in the last two phases. For the first fluidic input (phase Ip1, depicted in Fig. 2a), valves v_1, v_2, v_7 and v_8 are opened (together with v_4, v_5, v_6), the pump at the *Input* is activated and the liquid fills in the upper half of the mixer.

In the next phase Ip2, the second fluid sample fills the lower half of the mixer (Fig. 2c-i). Once both halves are filled, the mixer input and output valves (v_1 and v_8) are closed while valves v_2, v_3, v_7, v_9 are opened and the mixing operation is initiated (Fig. 2c-ii). Valve set $\{v_4, v_5, v_6\}$ acts as a peristaltic pump. Closing valve v_4 inserts some pressure on the fluid inside the mixer, closing valve v_5 creates further pressure, then as valve v_6 is closed valve v_4 is opened again. This forces the liquid to rotate clockwise in the mixer. The valves are closed and opened in a sequence such that the liquid rotates at a certain speed accomplishing the mixing operation. Next, in phase Op1 (Fig. 2c-iii), half of the mixed sample is pushed out of the mixer towards the rest of the chip and in Op2 (Fig. 2c-iv), the other half is transported to the waste.

Table 2 presents the control layer model of the pneumatic mixer shown in Fig. 2, whose flow layer model is characterized by the first row in Table 1. In Table 2, the valve activation for each phase is shown, '0' representing an open and '1' a closed valve. The status 'Mix' shown for the valve set $\{v_4, v_5, v_6\}$ on row 4 of Table 2 represents the mixing step in which these valves are opened and closed in a specific sequence to achieve mixing. The control layer model of a component contains all the details that a biochip controller requires for executing the operational phases of that component.

The different operational phases may or may not be executable in parallel depending on how the component is implemented, e.g., the mixer presented here has only one input port to receive both the input fluids, thus only one input phase can be activated at a time.

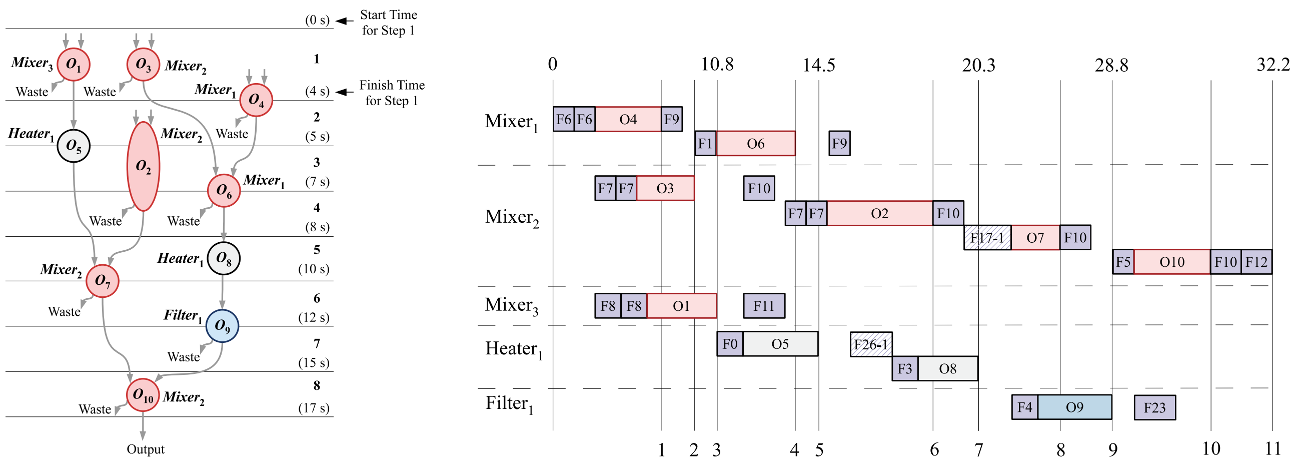
2.1.2 *Architecture Model*

We use our previously proposed topology graph-based model [14] in order to capture the biochip architecture. The biochip architecture shown in Fig. 4b is captured by $\mathcal{A} = (\mathcal{N}, \mathcal{S}, \mathcal{D}, \mathcal{F}, \mathcal{K}, c)$, where $\mathcal{N}$ is a finite set of vertices, $\mathcal{S}$ is a set of switches, $\mathcal{S} \subseteq \mathcal{N}$, $\mathcal{D}$ is a finite set of directed edges, $\mathcal{F}$ is a finite set of flow paths and $\mathcal{K}$ is a finite set of routing constraints. A vertex $N \in \mathcal{N}$ has two types: a vertex $S \in \mathcal{S}$ represents a switch (e.g., S_1 in Fig. 4b), whereas a vertex $M \in \mathcal{N}, \notin \mathcal{S}$, represents a component or an input/output node (e.g., $Mixer_1$ and In_1, respectively, in Fig. 4b).

The set of *flow paths* $\mathcal{F}$ is the set of permissible flow routes on the biochip. Each flow path has an associated control layer model that contains the details required for its utilization, i.e., the switch sequence and the pump activation details. A directed edge $D_{i,j} \in \mathcal{D}$ represents a directed communication channel from the vertex N_i to vertex N_j, with $N_i, N_j \in \mathcal{N}$. For example, in Fig. 4b, D_{Filter_1,S_5} represents a directed link from vertex $Filter_1$ to vertex S_5. A flow path, $F_i \in \mathcal{F}$, is either a single directed edge or a subset of two or more directed edges of $\mathcal{D}$, $F_i \subseteq \mathcal{D}$, representing a directed communication link between any two vertices $\in \mathcal{N}$. In Fig. 4b, $F_2 = (D_{Heater_1,S_{11}}, D_{S_{11},S_5}, D_{S_5,Mixer_2})$ represents a directed link from vertex $Heater_1$ to vertex $Mixer_2$. A routing constraint, $K_i \in \mathcal{K}$, is a set of flow paths that are mutually exclusive with the flow path $F_i \in \mathcal{F}$, i.e., none of the flow paths in the set can be activated in parallel. For example, F_2 and F_7 in Fig. 4b are mutually exclusive as they share the vertices S_5 and $Mixer_2$. The function $c(y)$, where y is either a directed edge $D \in \mathcal{D}$ or a flow path $F_i \in \mathcal{F}$, represents its routing latency (time required by a fluid sample to traverse y).

2.2 Biochemical Application Model

We model a biochemical application using a sequencing graph. The graph $\mathcal{G}(\mathcal{O}, \mathcal{E})$ is directed, acyclic and polar (i.e., there is a



(a) Allocation Example for Fig. 4a (b) Schedule

Figure 5: Illustrative Example

start vertex that has no predecessors and an *end* vertex that has no successors). Fig. 4a shows an example of a biochemical application. Each vertex $O_i \in \mathcal{O}$ represents an operation that can be bound to a component using a binding function $\mathcal{B} : \mathcal{O} \to M$. Each vertex has an associated weight $C_i(M_j)$, which denotes the execution time required for the operation O_i to be completed on component M_j. The execution times provided in Table 1 are of the actual functional phase (given in bold in the table, e.g., **Mix**). These execution times are taken as the typical execution times for the particular component types, i.e., typical mixing time is 0.5 s but a biochemical application description may specify a longer time (e.g., 5 s) if required for a certain operation. The edge set $\mathcal{E}$ models the dependency constraints in the assay, i.e., an edge $e_{i,j} \in \mathcal{E}$ from O_i to O_j indicates that the output of O_i is the input of O_j. An operation can start when all its inputs have arrived.

3. PROBLEM FORMULATION

The problem addressed in this paper can be formulated as follows: Given a biochemical application modeled as a sequencing graph $\mathcal{G}$ and a characterized component library $\mathcal{L}$, we are interested in synthesizing a biochip architecture $\mathcal{A}$ and a mapping Ψ of $\mathcal{G}$ onto $\mathcal{A}$, such that the application completion time is minimized and the imposed constraints are satisfied. The synthesis approach can handle several constraints, such as overall chip area, maximum number of components of a certain type and the external input and output ports. The number of external ports is also limited by the maximum number of punch holes possible on the chip under the given design rules [2]. As mentioned, the objective of the problem is to minimize the application completion time under the given constraints. However, other objectives can also be handled, such as the minimization of the architecture cost under a given timing constraint. The synthesis of the architecture $\mathcal{A}$ is the focus of this paper. The mapping of the application $\mathcal{G}$ onto the synthesized architecture is the focus of our previous work in [14].

Synthesizing an architecture $\mathcal{A}$ means deciding on (1) the allocation $\mathcal{U}$ of components from the component library $\mathcal{L}$, (2) the configuration for interconnection of these components (netlist), (3) placement $\mathcal{Z}_f$ of the components onto the chip layout area and interconnecting them by flow channel routing $\mathcal{R}_f$, and (4) placement $\mathcal{Z}_c$ of control valves and control ports on the chip and interconnecting them by control channel routing $\mathcal{R}_c$. The flow path set $\mathcal{F}$, associated latencies and the corresponding routing constraints $\mathcal{K}$ also need to be extracted from the synthesized architecture. These are given as an input to the application mapping stage.

Synthesizing a mapping $\Psi = <\mathcal{B}, \mathcal{X}>$ means deciding on (5) the binding $\mathcal{B}$ of the operations and edges in the application $\mathcal{G}$ onto the components and flow paths in the synthesized architecture $\mathcal{A}$ and (6) generating the corresponding schedule $\mathcal{X}$ while satisfying all the constraints imposed by the synthesized architecture, such as routing constraints.

Other constraints can also be catered for in the synthesis flow (using the same set of models) by including additional optimization steps. For example, reliability of an mLSI biochip depends directly on the reliability of the valves (the valves can operate reliably only up to a few thousand actuations). Therefore, in order to achieve enhanced reliability, an optimization step can be added directed at balancing the load on the valves, i.e., each valve goes through approximately the same number of valve actuations during the application execution.

4. BIOCHIP SYNTHESIS

The following subsections explain the design tasks involved in the biochip synthesis using Fig. 4 as an illustrative example. Section 5 presents our proposed synthesis framework for these tasks.

4.1 Allocation and Schematic Design

In this step, the microfluidic components required for implementing the given biochemical application $\mathcal{G}$ are allocated from the component library $\mathcal{L}$, while taking into account the imposed

Table 3: Allocated Components ($\mathcal{U}$)

Function	Constraints	Allocated Units	Notations
Input port	5	5	In_1 ... In_5
Output port	5	5	Out_1 ... Out_5
Mixer	3	3	$Mixer_1$ ... $Mixer_3$
Heater	2	1	$Heater_1$
Filter	1	1	$Filter_1$
Metering Units	3	3	Met_1 ... Met_3
Storage Units	4	4	$Storage_x$

resource constraints. Next, based on the given application, a chip schematic is designed and the netlist is generated. For example, to implement the biochemical application from Fig. 4a under the constraints given in Table 3 columns 1 and 2, we could use an allocation $\mathcal{U}$ as captured by the last two columns in Table 3. The schematic design corresponding to such an application and allocation is presented in Fig. 4b. Note that the storage units are needed in order to save the output of a component so that it can be used at a later stage. The flow path set is also generated in this step. A flow path is the path starting from the point of fluid sample origin and ending at the fluid sample destination point, e.g., $Heater_1$ to $Mixer_2$ in Fig. 4b. Source-sink paths associated with each flow path are also defined, e.g., for the flow path $Heater_1$ to $Mixer_2$ in Fig. 4b, the source-sink path is (In_4, S_{10}, $Heater_1$, S_{11}, S_5, $Mixer_2$, S_6, Out_2). Routing constraints are also extracted at this stage. Two flow paths, whose corresponding source-sink paths have a common vertex are mutually exclusive and need to be listed under the routing constraints, e.g., F_7 and F_2 in Fig. 4b are mutually exclusive since they share common vertices (e.g., S_5) in their source-sink paths. Table 5 shows the flow path set, the source-sink set and the routing constraints associated with the architecture in Fig. 4b. Additional routing constraints may be imposed during the placement and routing phases, resulting in an updated routing constraints list.

4.2 Physical Synthesis

In this step, the allocated components are placed on a chip layout area and the interconnections between components are routed as channels on the chip such that the application completion time is minimized. The placement and routing phases are governed by design rules (see Table 4) imposed by the fabrication process carried out in a standard microfluidic foundry [2, 4]. During placement, the components are treated as fixed size blocks, represented by rectangles, each having a fixed length and width. The placement is done in such a way that all design rules are satisfied and no two components overlap on the chip.

For mLSI-based biochips, the placement and routing phases can be divided into two stages, one for each logical layer in the chip: the flow layer and the control layer (in Fig. 1b, the flow layer is shown in blue and the control layer in red).

4.2.1 Flow Layer

This stage involves determining the placement of microfluidic components and the fluidic inlet/ outlet ports $\mathcal{Z}_f$ on the chip layout area, and then routing the interconnecting nets $\mathcal{R}_f$ as microfluidic flow channels. Only one layer is available for performing the flow channel routing [12]. In VLSI chips, the intersection of nets is considered a short-circuit and is thus not permitted. However, net intersection is possible in the biochip flow layer. A switch is placed at the location of the intersection so that both nets (a net represents a microfluidic channel) can be used, at different points in time, without unintended fluid mixing. Considering that only one layer is available for routing all flow channel nets, the possibility of net intersection helps in achieving 100% routability. However, net intersections cause routing constraints, resulting in longer application completion times. Fig. 4c shows the placement and routing scheme for the flow layer of the biochip architecture shown in Fig. 4b. The entire placement and routing shown is done in one layer.

4.2.2 Control Layer

In this stage, the placement of the control valves and the control ports $\mathcal{Z}_c$ is decided, and then the valves are connected to the control ports through control channel routing $\mathcal{R}_c$. In Fig. 1b, the control layer is shown in red, with the control valves labelled as v_x and the control ports as z_y. Positions of the valves that are used inside a microfluidic component can be obtained directly from the component library. The positions of the valves that need to be placed on the flow channels are inferred from the flow routing information (e.g., valves need to be placed at all flow channel intersections). As explained in Section 2, one logical control layer can have two physical layers that can be used for placement and routing (above and below the flow layer) [12]. Contrary to the flow channels, control channels are not allowed to intersect.

Table 4: Design Rules

Parameter	Suggested Value
Width of flow channel	100 μm
Minimum spacing between flow channels	40 μm
Width of control channel	30 μm
Width of control valve	100 μm
Minimum spacing between control channels	40 μm
Minimum spacing between external ports	1500 μm

After the placement is complete, the next step is to connect the valves to the ports using control channels. The control channels can be routed over/ under any flow channel/ component without forming a valve. The crossing of the control channel over a flow channel forms a valve only if the control channel has a large width (100 μm) [2]. The flow path channel lengths (used to calculate the routing latencies) and any additional routing constraints (imposed because of net intersections in the flow layer) can now be extracted from the layout and captured in the biochip architecture model $\mathcal{A}$. Table 5 shows the routing constraints and the list of flow paths for the biochip architecture in Fig. 4b together with their corresponding routing latencies.

4.3 Application Mapping

The next step is mapping the biochemical application $\mathcal{G}$ onto the synthesized architecture $\mathcal{A}$ such that the application completion time is minimized and the dependency, resource and routing constraints are satisfied. The binding $\mathcal{B}$ for the operations is the same as determined when generating the schematic. Binding of the edges and scheduling $\mathcal{X}$ for both the operations and the edges is generated in this step. Fig. 5b shows the schedule for the case when the application in Fig. 4a is scheduled on the architecture in Fig. 4b. The schedule is represented as a Gantt chart, where, we represent the operations and fluid routing phases as rectangles, with their lengths corresponding to their execution duration.

5. SYNTHESIS STRATEGY

Fig. 6 shows the block diagram of our proposed design methodology. In this paper, we focus on the "Architectural Synthesis" block which synthesizes the biochip architecture which is then given as input to the "Application Mapping" block. For the application mapping, we use our previously proposed approach in [14]. Microfluidic platforms are equipped with a controller that manages all on-chip control, i.e., issuing signals to on-chip components for executing a biochemical application, performing data acquisition and signal processing operations [10]. The mapping implementation (containing the binding and scheduling information), together with the component and biochip architecture models, can be used to generate the control sequence for a biochip controller ("Control Synthesis") in order to automatically execute the biochemical application onto the synthesized biochip.

The biochip synthesis problem presented in Section 3 is NP-complete (the scheduling step, even in simpler contexts is NP-com-

Table 5: Flow Path Set ($\mathcal{F}$), Source-Sink Set and Routing Constraints ($\mathcal{K}$)

Flow Path Set	Source-Sink Set	Routing Constraints
$F_0 = (Mixer_3, S_9, S_{10}, Heater_1)$, 0.7 s	$F'_0 = (In_3, Met_3, S_7, S_8, Mixer_3, S_9, S_{10}, Heater_1, S_{11},$	$K_0 : (F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8,$
$F_1 = (Mixer_2, S_6, S_2, Mixer_1)$, 0.4 s	$Out_2)$	$F_9, F_{10}, F_{11}, F_{12}, F_{13}, F_{14}$
$F_2 = (Heater_1, S_{11}, S_5, Mixer_2)$, 0.5 s	$F'_1 = (In_2, Met_2, S_4, S_5, Mixer_2, S_6, S_2, Mixer_1, S_3,$	$F_{15}, F_{16}, F_{17}, F_{18}, F_{19}, F_{20},$
$F_3 = (Mixer_1, S_3, S_{10}, Heater_1)$, 0.6 s	$Out_1)$	$F_{21}, F_{24}, F_{25}, F_{26}, F_{27}, F_{28})$
$F_4 = (Heater_1, S_{11}, S_{12}, Filter_1)$, 2.1 s	$F'_2 = (In_4, S_{10}, Heater_1, S_{11}, S_5, Mixer_2, S_6, Out_2)$	K_1: $(F_0, F_2, F_3, F_4, F_5, F_6, F_7, F_8,$
$F_5 = (Filter_1, S_{13}, S_5, Mixer_2)$, 0.8 s	$F'_3 = (In_1, S_{16}, Met_1. S_1, S_2, Mixer_1, S_3, S_{10}, Heater_1,$	$F_9, F_{10}, F_{12}, F_{13}, F_{14}, F_{15}, F_{16},$
$F_6 = (In_1, S_{16}, Met_1, S_1, S_2, Mixer_1)$, 1.3 s	$S_{11}, Out_2)$	$F_{17}, F_{24}, F_{25}, F_{26})$
$F_7 = (In_2, Met_2, S_4, S_5,Mixer_2)$, 1.9 s	$F'_4 = (In_4, S_{10}, Heater_1, S_{11}, S_{12}, Filter_1, Out_3)$	...
$F_8 = (In_3, Met_3, S_7, S_8, Mixer_3)$, 2.1 s	$F'_5 = (In_1, S_{16}, S_{12}, Filter_1, S_{13}, S_5, Mixer_2, S_6, Out_2)$	...
$F_9 = (Mixer_1, S_3, Out_1)$, 1.2 s	$F'_6 = (In_1, S_{16}, Met_1, S_1, S_2, Mixer_1, S_3, Out_1)$	K_{28-x}: $(F_0, F_3, F_4, F_5, F_6, F_7, F_9,$
$F_{10} = (Mixer_2, S_6, Out_2)$, 0.3 s	$F'_7 = (In_2, Met_2, S_4, S_5,Mixer_2, S_6, Out_2)$	$F_{11}, F_{14}, F_{15}, F_{16}, F_{17},$
$F_{11} = (Mixer_3, S_9, Out_3)$, 0.6 s	...	$F_{18}, F_{19}, F_{20}, F_{21}, F_{22},$
...	$F'_{28-x} = (In_1, S_{16}, S_{12}, Filter_1, S_{13}, S_{14}, Storage, S_{15},$	$F_{23}, F_{25}, F_{26}, F_{27})$
$F_{28-x} = (Filter_1, S_{13}, S_{14}, Storage)$, 0.5 s	$Out_5)$	

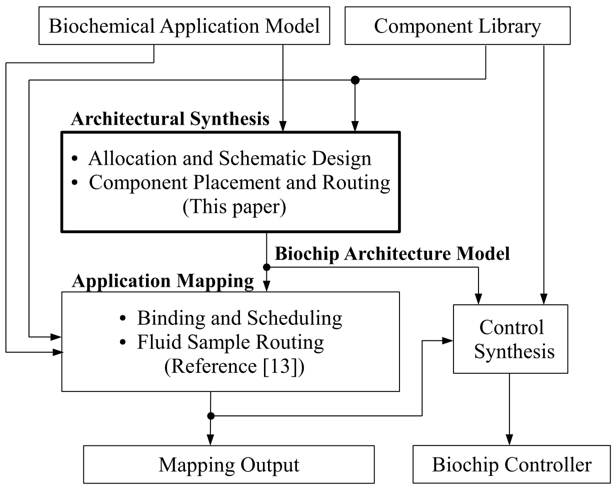


Figure 6: Design Methodology

plete [18]). Our synthesis strategy in this paper is to solve each design task separately, by adapting well-known heuristic algorithms from VLSI domain. The heuristics do not guarantee obtaining the optimal solution. Obtaining the optimal results (in terms of application completion time) is infeasible even for small examples. The following subsections present the chosen heuristics and describe our strategy using Fig. 4 as an example.

5.1 Allocation and Schematic Design

This stage receives the application graph $\mathcal{G}$, component library $\mathcal{L}$ and the resource constraints as input and determines the allocation $\mathcal{U}$ and generates the netlist. All components of the architecture model $\mathcal{A}$ are captured here except the routing latency c.

5.1.1 Allocation

The most common approach for allocation in High-Level Synthesis (HLS) [8] is to use resource-constrained List-Scheduling and binding. We start off by topologically sorting the operations of the biochemical application based on their dependency constraints and then prioritizing them using an *urgency criteria* [8]. The urgency of an operation is specified by the length of the longest path from the operation to the end node in the application graph. An operation is considered *ready*, if all of its predecessors have finished execution. All the operations in the application are evaluated, the ready ones are found and are placed in a ready list *RL*. For example in Fig. 4a, operations O_1 to O_4 have no predecessors and are thus considered ready, whereas O_6 cannot be executed until O_3 and O_4 are complete. For each ready operation we allocate a component of the required type, considering the imposed constraints (see Table 3, column 2). The operation is bound greedily to the allocated component and scheduled.

Fig. 5a shows the allocation schedule for the application in Fig. 4a. The schedule is divided into 8 schedule steps. The start of an operation marks the start of a schedule step (O_1, O_3, O_4 start at time t = 0 s, thus starting schedule step 1) and an operation completion marks the end of a schedule step (schedule step 1 ends at 4 s as operations O_1 and O_3 finish, and schedule step 2 ends at 5 s when operation O_4 finishes). Unlike the control steps in HLS [8] (where all control steps represent a fixed time duration, a clock cycle), the schedule steps are of varying time lengths.

The binding of each operation is also shown in Fig. 5a, the component name is placed next to the operation (e.g., O_1 is bound to $Mixer_3$). If the number of ready operations exceeds the number of available components, then the least urgent operations (i.e., greedy binding based on the urgency criteria) are deferred, e.g., in Fig. 5a, O_2 is deferred to schedule step 2 as there are only three mixers available for usage in schedule step 1.

As soon as an operation completes, it marks the end of a schedule step. The operations are then re-evaluated to find the new list of ready operations and the process is repeated. Table 3 shows the list of allocated components. All imposed resource constraints have been fulfilled. All components have been used in their maximum allowed number except the heater. Only one heater has been allocated considering the requirement, compared to the 2 heater units that were allowed by the user. At this stage, the routing latencies are not yet known. The actual values of the routing latency are generated after the placement and routing is complete. A set of input and output ports is allocated for each component in order to serve as the source and sink point during flow path execution. Metering units are used to create discretized samples of unit volume. The number of metering units allocated depends on the maximum possible external inputs that can be executed in parallel. In the current case, a maximum of three external inputs are taken in parallel in schedule step 1 (Fig. 5a) and thus three metering units have been allocated.

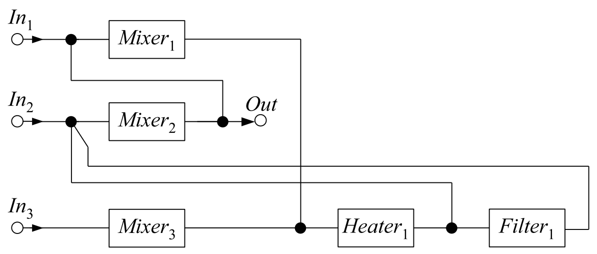


Figure 7: Schematic

5.1.2 Schematic Design

In the next step, we extract the schematic design from the generated binding and scheduling information (analogous to data path generation in HLS [8]). Each schedule step (Fig. 5a) is scanned to find the input source of the utilized components and a corresponding net is placed between the component and the input source. For example, $Heater_1$ in schedule step 5 gets an input from $Mixer_1$ and gives an output to $Filter_1$, therefore it is connected to both components. The extracted component interconnection scheme is shown in Fig. 7.

Next we connect input and output ports to each component to serve as the source and sink point. Metering units are placed at the fluidic sample input ports. Storage units are needed to store intermediate results of operations [19]. Unlike in HLS (where registers are required at every control step [8]), biochips require storage only under special conditions. Consider an operation O_x bound to a component M_y that finishes execution. If another operation gets bound to M_y in the next schedule step and the successor operation of O_x has not been scheduled yet, then the output of O_x will need to be moved to the storage. Each storage unit is capable of storing multiple fluid unit samples, depending on the number of storage channels inside the unit. Since the routing latencies are not yet known, it is not possible to accurately assess which components would require the storage unit usage. For now, the storage unit is is connected to all components and the designer specifies the maximum capacity of the unit. Unnecessary connections and extra storage channels are removed after the application mapping step. The final component interconnection configuration is shown in Fig. 4b.

The flow path set $\mathcal{F}$ and the corresponding routing constraints $\mathcal{K}$ are also generated in this step. The flow path set for the biochip architecture in Fig. 4 is shown in Table 5. The source-sink path for each flow path and the routing constraints are also shown.

5.2 Physical Synthesis

This stage takes the allocation $\mathcal{U}$, the netlist, component library $\mathcal{L}$ and the desired layout size as input and performs placement and routing ($\mathcal{Z}_f$, $\mathcal{R}_f$, $\mathcal{Z}_c$, $\mathcal{R}_c$), determining any additional constraints in the set $\mathcal{K}$ and the routing lengths of the flow paths (that are used to calculate the routing latencies c). We use a grid-based approach to perform physical synthesis. The grid size is dictated by the design rules and the component sizes on the grid are calculated accordingly using their dimensions given in the component library. The design rules imposed by the foundry and followed during the physical synthesis are summarized in Table 4. The placement and routing phases are divided into two stages, one for each logical layer.

5.2.1 Flow Layer

According to the problem formulation from Section 3, the placement and routing should be done such that the application completion time is minimized. However, this would require using the mapping and scheduling (Section 5.3) as the cost function, which is too time-consuming. Instead, we use the total channel length and the number of net intersections as the cost function. Minimizing the channel length minimizes the routing latencies, in turn minimizing the application completion time. Similarly, minimizing the number of net intersection minimizes the number of routing constraints, allowing more flow paths to be executed in parallel. This will not lead to the optimal result, but will reduce the application completion time. Furthermore, performing actual routing to compare various placement solutions is impractical as routing is a time-consuming process. Therefore, we perform placement and routing in separate steps and use an estimation method (Semi-Perimeter method, the most widely used approximation method [16]) to estimate the total channel length in order to judge the quality of the placement solution.

The placement of components such that the total channel length is minimized is an NP-complete problem, for which a number of good heuristic techniques have been developed [16]. Considering the problem at hand, we use Simulated Annealing (SA) (one of the most used methods for cell placement in VLSI [16]) for performing component placement on the chip. Various algorithms have been proposed for routing over the years. We use Hadlock's Algorithm (HA) [16] and extend it for the flow layer routing. HA is suitable for the current problem since it also uses a grid model approach, finds the shortest path between two vertices (if such a path exists) and is faster than the other algorithms in this category [16]. We extend HA to also consider the possibility of net intersections, ensuring a 100% routability. The quality of the solution is judged by the number of net intersections and the total channel length. Since HA is sensitive to the order in which the nets are routed, we iterate on HA, providing it a re-ordered netlist for every iteration in order to achieve a routing solution that minimizes the total channel length and net intersections.

The routing latency corresponding to each flow path is also generated in this step. Routing latencies are calculated by using the routing length of each flow path extracted from the architecture and the flow rate used on the chip. We consider a flow rate of 10 mm/s for all experiments in this paper. If the flow path length is 10 mm and the flow rate is set at 10 mm/s, then a unit volume of liquid (10 mm length on the channel) traverses this flow path in 2 s, i.e., from the time the tip of the 10 mm unit sample enters the flow path till the time the tail leaves from it. The latency values are required while performing application mapping. Latency values generated for each flow path are shown in Table 5.

Physical Synthesis Algorithm. Fig. 8 shows our algorithm for the physical synthesis of the flow layer. The algorithm takes the allocated component set *M*, the generated netlist *List* and the component library $\mathcal{L}$ as an input, and returns the placement and routing information of the flow layer. The objective is to place all the components on the chip and minimize the total channel length in order to reduce the routing latencies, while satisfying the design rules. Fig. 4c shows the flow layer placement and routing scheme that comes out of our algorithm.

Simulated Annealing [16] (lines 1−17 in Fig. 8) is used for generating the placement scheme. SA is a metaheuristic, which, starting from a random initial placement of components (line 3), iteratively obtains a better placement scheme by performing *moves* (line 6), i.e., design transformations, (swapping, rotating or randomly changing component location on the chip) to modify the current solution. SA also accepts deteriorations in cost (lines 11−13) to a limited extent in an effort to obtain global optimum, in terms of the cost function used. The placement generated by SA $\mathcal{Z}_f$ is given as

FlowPlaceAndRoute(M, $List$, $\mathcal{L}$)

```
1  // Phase I: Flow Layer Placement
2  Initialize $T$
3  $\mathcal{Z}_f^{now}$ = InitialPlacement($M$, $\mathcal{L}$)
4  repeat
5      for i=1 to $TL$ do
6          $\mathcal{Z}'_f$ = moves($\mathcal{Z}_f^{now}$)
7          $\delta$ = cost($\mathcal{Z}'_f$) $-$ cost($\mathcal{Z}_f^{now}$)
8          if $\delta < 0$ then
9              $\mathcal{Z}_f^{now} = \mathcal{Z}'_f$
10         else
11             if $random(0,1) < e^{-\delta/T}$ then
12                 $\mathcal{Z}_f^{now} = \mathcal{Z}'_f$
13             end if
14         end if
15     end for
16     $T = \alpha \times T$
17 until <stop criterion is met>
18 // Placement returns the best solution $\mathcal{Z}_f$
19 // Phase II: Flow Channel Routing
20 $\mathcal{R}_f^{now}$ = RouteFlowLayer($\mathcal{Z}_f$, $List$)
21 $c^{now}$ = cost($\mathcal{R}_f^{now}$)
22 repeat
23     $List$ = Re-order($List$)
24     $\mathcal{R}'_f$ = RouteFlowLayer($\mathcal{Z}_f$, $List$)
25     $c'$ = cost($\mathcal{R}'_f$)
26     if $c' < c^{now}$ then
27         $c^{now} = c'$
28         $\mathcal{R}_f^{now} = \mathcal{R}'_f$
29     end if
30 until <stop criterion is met>
31 return <$\mathcal{Z}_f$, $\mathcal{R}_f^{now}$>
```

Figure 8: Physical synthesis algorithm for the flow layer

input to the Hadlock's Algorithm [16] (lines 20−30) to iteratively generate the routing. A re-ordered netlist is generated (line 23) in every iteration in order to cater for HA's sensitivity to the order in which the nets are routed. The best solution for the placement $\mathcal{Z}_f$ and the routing $\mathcal{R}_f^{now}$ is then returned (line 31).

5.2.2 Control Layer

Control layer placement and routing can be done using the same algorithms as described for the flow layer. We aim to target this step in our future research. Since the number of control valves on these chips can be extremely high (commercial chip having more than 25,000 valves [15]) and the number of punch holes that can be made on the chip for connecting the control ports is limited by the design rules [2], each valve cannot be connected to a separate control port. Different approaches aimed at sharing the control ports between valves have been proposed [12], which can be used to reduce the required number of control ports.

5.3 Application Mapping

Now since we have the biochip architecture $\mathcal{A}$, the biochemical application $\mathcal{G}$ and the characterized component library $\mathcal{L}$, we use our previously proposed binding and scheduling strategy [14] for mapping the given application onto the synthesized architecture such that the application completion time is minimized and all the imposed constraints are satisfied. Fig. 5b shows the schedule determined by our tool when the application in Fig. 4a is scheduled on the architecture in Fig. 4b. We use the same binding as the one generated during the schematic design (Fig. 5a). As shown in Fig. 5b, the application requires only one storage reservoir. The output of operation O_5 is moved to the storage since the heater needs to be reused for operation O_8 before the successor operation of O_5 (which is operation O_7) is released. The application execution is completed in 32.2 s.

6. EXPERIMENTAL EVALUATION

We evaluate our proposed approach by synthesizing biochip architectures for three real life assays and a set of four synthetic benchmarks. We implement these applications onto the synthesized chips and determine the application completion time. The algorithm was implemented in C#, running on a Pavilion laptop (HP dv6-2155dx) with Core i3, Dual Processors at 2.13 GHz and 4 GB of RAM.

Table 6 shows our experimental results. Column 1 presents the application and column 2 shows the list of allocated components, in the following format: (Input ports, Output ports, Mixers, Heaters, Filters, Detectors). Columns 3−6 present the desired chip area, total length of the flow channels, total number of net intersections and the total number of valves on the chip, respectively. Chip area represents the area given by the user as input. Chip area and total channel lengths are scaled, with a unit length being equal to 150 μm, i.e., a total length of 10 given in Table 6 corresponds to 1500 μm. The number of valves are calculated by considering 1 valve for each I/O port, 4 valves for each intersection (switch), 9 valves for each mixer, 6 valves for each metering unit and 2 valves each for all remaining components. This valve count can be further minimized by removing the valves which are never used. We plan to target this in our future research. The last column presents the completion time $\delta_{\mathcal{G}}$ of the application, in seconds, on the synthesized architecture.

Real-life assays can be converted to our application model using [5]. The first real-life assay we use is the PCR (polymerase chain reaction) mixing stage that has 7 mixing operations and is used in DNA amplification. The architecture details and the corresponding application completion time are shown in row 1 of Table 6. Row 2 shows the architecture generated for Multiplexed IVD (in-vitro diagnostics) that has a total of 12 operations and is used to test different fluid samples from the human body. The third row shows a larger real-life application, a colorimetric protein assay (CPA, 55 operations), utilized for measuring the concentration of a protein in a solution. It uses a chip equipped with 295 valves to complete its execution in 72.7 s. The architectural details given in row 4 are for the example application (EA) given in Fig. 4a.

In the second set of experiments we have evaluated our proposed method using a set of four synthetic benchmarks. The benchmark applications are composed of 10, 30, 40 and 50 operations. Table 7 shows the details of the synthesized architectures considered and the respective application completion times achieved.

For each application in Table 7, two sets of architectures were synthesized. The first row presents results for the architecture syn-

Table 6: Real-Life Applications

Appl.	Allocated Units	Chip Area	Net Length	Total Inters.	Total Valves	$\delta_{\mathcal{G}}$
PCR	(3, 3, 3, 0, 0, 0)	250 × 250	198	4	67	19.7 s
IVD	(5, 5, 3, 0, 0, 3)	250 × 250	393	10	101	20 s
CPA	(5, 5, 5, 0, 0, 3)	250 × 250	1360	51	295	72.7 s
EA	(5, 5, 3, 1, 1, 0)	150 × 150	1917	63	311	32.2 s

Table 7: Synthetic Benchmarks

$\|O\|$	Allocated Units	Chip Area	Net Length	Total Inters.	Total Valves	δ_G
10	(2, 2, 1, 1, 1, 1)	150 × 150	1813	45	211	39.9 s
	(5, 5, 2, 1, 1, 1)	150 × 150	1926	68	324	35.7 s
30	(6, 6, 3, 2, 2, 1)	250 × 250	3575	122	573	64.5 s
	(15, 18, 6, 4, 3, 1)	350 × 350	5243	124	665	46.1 s
40	(8, 8, 4, 3, 1, 2)	350 × 350	4799	151	716	69.8 s
	(18, 20, 7, 5, 2, 3)	350 × 350	7452	171	889	59.5 s
50	(10, 10, 5, 2, 2, 2)	350 × 350	6522	177	839	81.25 s
	(21, 24, 10, 4, 3, 3)	400 × 400	9366	213	1109	60.1 s

thesized under designer-given constraints (maximum number of components of a certain type is constrained), whereas, the second row presents the results of an unconstrained architecture, i.e., no constraints were placed on the number of components to be used. Allocation step for the unconstrained architecture case can be considered similar to ASAP Scheduling [13]. For all applications, the unconstrained architecture produces a completion time that is smaller than that of the constrained architecture. All experiments presented in this section took between 3 to 30 minutes to complete, depending on the complexity of the application. All benchmarks and test files can be found here [3].

7. CONCLUSIONS

In this paper we have presented a top-down architectural synthesis approach for flow-based microfluidic biochips. The proposed approach synthesizes a biochip architecture for a given biochemical application, such that the application completion time is minimized. The synthesis process involves component allocation, design schematic generation, and the physical synthesis (placement and routing) of the chip. The approach has been evaluated by synthesizing biochip architectures for three real-life assays and a set of synthetic benchmarks. To the best of our knowledge, this is the first time an architectural synthesis framework has been proposed for the mLSI biochips. The proposed approach is expected to facilitate programmability and automation in the microfluidics domain, reducing human effort and minimizing the design cycle time.

8. ACKNOWLEDGMENTS

This work was supported by the Danish Agency for Science, Technology and Innovation, Grant No. 2106-08-0018 “ProCell”.

9. REFERENCES

[1] AutoCAD Products. `http://usa.autodesk.com/autocad-products/`.

[2] Designing your own device: Basic design rules. `http://www.stanford.edu/group/foundry/`.

[3] mLSI Biochips. `https://sites.google.com/site/mlsibiochips/`.

[4] N. Amin, W. Thies, and S. Amarasinghe. Computer-aided design for microfluidic chips based on multilayer soft lithography. In *Proceedings of the IEEE International Conference on Computer Design*, 2009.

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

[6] K. Chakrabarty and T. Xu. *Digital Microfluidic Biochips: Design Automation and Optimization*. CRC Press, Boca Raton, FL, 2010.

[7] H. Chou, M. Unger, and S. Quake. A microfabricated rotary pump. *Biomedical Microdevices*, 3, 2001.

[8] P. Coussy and A. Morawiec. *High-level synthesis: From algorithm to digital circuit*. Springer, 2008.

[9] J. W. Hong and S. R. Quake. Integrated nanoliter systems. *Nature Biotechnology*, 21:1179–1183, 2003.

[10] Y. C. Lim, A. Z. Kouzani, and W. Duan. Lab-on-a-chip: a component view. *Journal of microsystems technology*, 16(12), December 2010.

[11] D. Mark, S. Haeberle, G. Roth, F. von Stetten, and R. Zengerle. Microfluidic lab-on-a-chip platforms: requirements, characteristics and applications. *Chem. Soc. Rev.*, 39:1153–1182, 2010.

[12] J. Melin and S. Quake. Microfluidic large-scale integration: The evolution of design rules for biological automation. *Annual Reviews in Biophysics and Biomolecular Structure*, 36:213–231, 2007.

[13] G. D. Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, New York, 1994.

[14] W. H. Minhass, P. Pop, and J. Madsen. System-level modeling and synthesis of flow-based microfluidic biochips. In *Proc. of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems (CASES)*, 2011.

[15] J. M. Perkel. Microfluidics - bringing new things to life science. *Science*, November 2008.

[16] S. M. Sait and H. Youssef. *VLSI physical design automation: theory and practice*. World Scientific Publishing Co. Pte. Ltd., 1999.

[17] J. Siegrist, M. Amasia, N. Singh, D. Banerjee, and M. Madou. Numerical modeling and experimental validation of uniform microchamber filling in centrifugal microfluidics. *Lab Chip*, 10:876–886, 2010.

[18] D. Ullman. NP-complete scheduling problems. *Journal of Computing System Science*, 10:384–393, 1975.

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