Contamination-Free Switch Design and Synthesis for Microfluidic Large-Scale Integration

Duan Shen, Yushen Zhang, Mengchu Li, Tsun-Ming Tseng, and Ulf Schlichtmann
Chair of Electronic Design Automation, Technical University of Munich, Arcisstrasse 21, D-80333 Munich, Germany
{duan.shen, yushen.zhang, mengchu.li, tsun-ming.tseng, ulf.schlichtmann}@tum.de

***Abstract*—Microfluidic large-scale integration (mLSI) biochips have developed rapidly in recent decades. The gap between design efficiency and application complexity has led to a growing interest in mLSI design automation. The state-of-the-art design automation tools for mLSI focus on the simultaneous co-optimisation of the flow and control layers but neglect potential contamination between different fluid reagents and products. Microfluidic switches, as fluid routers at the intersection of flow paths, are especially prone to contamination. State-of-the-art tools design the switches as spines with junctions, which aggregate the contamination problem. In this work, we present a contamination-free microfluidic switch design and a synthesis method to generate application-specific switches that can be employed by physical design tools for mLSI. We also propose a scheduling and binding method to transport the fluids with least time and fewest resources. To reduce the number of pressure inlets, we consider pressure sharing between valves within the switch. Experimental results demonstrate that our methods show advantages in avoiding contamination and improving transportation efficiency over conventional methods.**

***Index Terms*—microfluidic large-scale integration, design automation, quadratic linear programming, contamination**

I. Introduction

Over the last few decades, microfluidic large-scale integration (mLSI) has gained increasing interest in the fields of biology and chemistry. Manually designing mLSI chips is a time-consuming and error-prone procedure. Design automation for mLSI thus arose to alleviate design difficulty and to enhance design quality. In particular, the state-of-the-art physical synthesis tool Columba has demonstrated the ability to synthesise manufacturing-ready mLSI designs within a few minutes or seconds [1]–[3]. Despite the advances, current mLSI design automation tools generally focus on conventional physical design features such as area reduction, valve and inlet reduction, etc, but neglect practical bio-constraints such as the inter-contamination among fluids.

Contamination is a critical issue for mLSI. When a fluid flows through a channel, residues may be left over in the channel. These residues may contaminate subsequent fluids that pass through the same channel. Because bioassays are operated with significantly reduced amounts of fluids, the concentration of the contaminated fluids can be amplified, resulting in erroneous experimental results. For example, in a PCR reaction, if different DNA samples flow through the same channel segment, residues left by the former sample can contaminate the latter sample, leading to failure of the experiments [4].

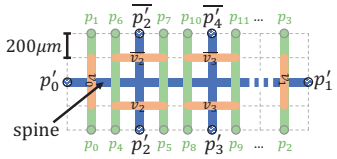

Fig. 1. A switch design applied in Columba. The blue, green and orange lines represent the flow channels, control channels and valves, respectively.

Current mLSI design methods tend to reduce the number of fluid transportation channels to minimise the chip area. The reduction of channels however requires more fluids to share the same channel segments, which further aggravates the contamination problem. In particular, contamination can be severe at microfluidic switches. Switches are mLSI-components to guide fluids at the intersections of flow channels. In mLSI designs synthesised by state-of-the-art tools, switches are designed as a spine with many junctions, as shown in Fig. 1. All fluids entering the switches will pass the spine and thus may contaminate the subsequent fluids.

To deal with the contamination problem, some works proposed to incorporate rinsing operations into the assay to remove the residues [4], [5]. These methods can safely avoid contamination but will also prolong the assay execution process. Other works proposed contamination-free microfluidic designs for specific bioassays [6], but the designs cannot be generalised to support an arbitrary bio-application.

In this paper, we propose a contamination-free microfluidic switch design which can be employed by automatic physical synthesis tools for mLSI. The advantages of our method over state-of-the-art methods are three-fold. First, our method prevents fluid contamination without increasing the assay execution time; second, our method supports concurrent transportation of multiple fluids to improve the transportation efficiency; third, our method reduces the number of valves and pressure inlets required for the fluid transportation.

II. Contamination-free Switch Design

Many bio-applications involve fluids that must not be mixed with one another, such as the reagents used for different experimental and control groups. These fluids need to be treated in different devices [7] and should not be transported using the same channels unless we rinse the channels after each transportation [4]. In this paper, we refer to two fluids that must not be mixed as a pair of *conflicting fluids*.

Besides conflicting fluids, there are also fluids that are not prone to contamination, e.g. dilution buffers that do not react with other fluids. These fluids can safely be transported in a flow channel that has been or will be used by other fluids. Nevertheless, different fluids cannot be transported using the same flow channel segment at the same time.

State-of-the-art switch designs have a spine-based structure, i.e. fluids entering the switch need to travel along the "spine" channel before arriving at their destined junctions. Such designs have two problems: 1) when a pair of conflicting fluids need to be transported using the same switch, the former fluid may leave residues in the spine and thus contaminate the latter fluid; and 2) multiple fluids cannot use the switch at the same time since they all need to access the "spine" channel.

Targeting the present deficiencies, we propose a novel switch design which arranges the flow channel segments in a distributive manner to avoid unnecessary overlapping of different transportation paths. Fig. 2 depicts our design for an 8-pin switch which consists of up to 8 flow pins and a 12-pin switch that has up to 12 flow pins. All channels and valves in the designs can be considered as placeholders that can be synthesised on demand. Specifically, our design can be modelled as:

- a set $\mathcal{P}$ of flow channel pins via which fluids can enter and leave the switch, e.g. for the 8-pin switch, $\mathcal{P} := \{T1, T2, R1, R2, B1, B2, L1, L2\}$;
- a set $\mathcal{N}$ of intermediate nodes representing the intersections of flow channels inside the switch, e.g. for the 8-pin switch, $\mathcal{N} := \{TL, T, TR, L, C, R, BL, B, BR\}$; and
- a set $\mathcal{S}$ of flow channel segments between two nodes or between a node and a flow pin, e.g. for the 8-pin switch, $(L1, L) \in \mathcal{S}$.

In this manner, we can model a fluid transportation path f^p inside the switch as a sequence of flow channel segments. E.g. $f^p = [(L1, L), (L, TL), (TL, T1)]$ represents a path consisting of three channel segments to transport the fluids from the left to the top of the switch. In particular, we design the same number of flow pins on every boundary of the switch, so that fluids can access and leave the switch from all directions. Since fluid transportation no longer relies on a single "spine" channel in our switch design, we are able to support multiple fluid transportation paths without overlapped flow channel segments. Thus, conflicting fluids can safely pass the switch without contamination, and multiple fluids can use the switch at the same time without collision.

To note is that in most cases, we do not need to activate all placeholders in the design. For example, if we want to support two fluid transportation paths from the left to the top and from the left to the bottom of the switch using the 8-pin design, we just need to synthesise a subset of the switch components, e.g. $P' = \{L1, T1, L2, B1\}$, $\mathcal{N}' = \{TL, L, BL, B\}$ and $\mathcal{S}' = \{(L1, L), (L, TL), (TL, T1), (L2, BL), (BL, B), (B, B1)\}$. All other components are not necessary for the transportation and thus do not need to be fabricated in the final design.

In order to transport all fluids with fewest possible resources and least possible time, we propose an integer-quadratic programming (IQP) model to automatically synthesise necessary switch components, schedule the fluid transportation operations, and bind the flow pins of the switch to other on-chip devices involved in the transportation.

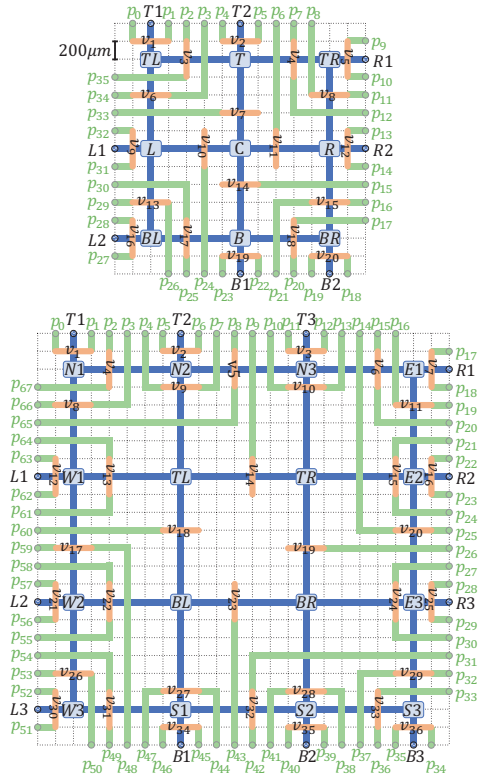

Fig. 2. Switch designs with 8 and 12 pins. The blue, green and orange lines represent the flow channels, control channels and valves, respectively.

III. Mathematical Model

Based on the fluid transportation requirements, we can choose either the 8-pin or the 12-pin switch design as the starting point of the synthesis. Besides the notations $\mathcal{P}$, $\mathcal{N}$ and $\mathcal{S}$ introduced in section II, we introduce a set $\mathcal{F}$ containing all the fluids that need to be transported, and a set $\mathcal{F}_C \subseteq \mathcal{F} \times \mathcal{F}$ containing all pairs of conflicting fluids.

A. Preparation: Collecting Fluid Transportation Options

To avoid detours of the transportation paths, before building the mathematical models, we first find out all shortest paths between every two flow pins in the switch with Dijkstra's algorithm and collect these shortest paths in a set $\mathbb{P}$ as candidate paths for fluid transportation. E.g. for an 8-pin switch, $f^p = [(L1, L), (L, TL), (TL, T1)] \in \mathbb{P}$ since f^p is the shortest path between flow pins $L1$ and $T1$.

B. Contamination Avoidance

For each to-be-transported fluid, we arrange a transportation path by introducing a binary variable x_{f,f^p} for each $f \in \mathcal{F}$

and $f^p \in \mathbb{P}$ to represent whether fluid f is transported via path f^p. This can be modelled with the following constraints:

$$\sum_{f^p \in \mathbb{P}} x_{f,f^p} = 1, \forall f \in \mathcal{F}. \tag{1}$$

To avoid contamination, the transportation paths of two conflicting fluids should not contain the same intermediate node. To this end, we introduce a binary constant $b_{f^p,n}$ for all candidate paths $f^p \in \mathbb{P}$ and intermediate nodes $n \in \mathcal{N}$ to represent whether path $f^p \in \mathbb{P}$ contains node n. We then introduce the following constraints:

$$\sum_{f^p \in \mathbb{P}} x_{f_1,f^p} \cdot b_{f^p,n} + \sum_{f^p \in \mathbb{P}} x_{f_2,f^p} \cdot b_{f^p,n} \leq 1, \quad \forall n \in \mathcal{N}, (f_1, f_2) \in \mathcal{F}_C. \tag{2}$$

Thus, for a pair of conflicting fluids (f_1, f_2), if f_1 is bound to any path containing node n, i.e. there exists $f^{p'} \in \mathbb{P}$ so that $x_{f_1,f^{p'}} = 1$ and $b_{f^{p'},n} = 1$, $\sum_{f^p \in \mathbb{P}} x_{f_1,f^p} \cdot b_{f^p,n}$ will be equal to 1. In this case, constraint (2) can only be fulfilled when $\sum_{f^p \in \mathbb{P}} x_{f_2,f^p} \cdot b_{f^p,n} = 0$, i.e. for all candidate paths $f^p \in \mathbb{P}$, either f^p is not bound by f_2, i.e. $x_{f_2,f^p} = 0$, or f^p does not contain node n, i.e. $b_{f^p,n} = 0$.

C. Scheduling

Our switch design supports concurrent transportation of multiple fluids under the prerequisite that the corresponding transportation paths do not overlap. We model this as a scheduling problem. Specifically, we assign each fluid to a time slot for transportation. A time slot can accommodate multiple fluids, but different fluids that are transported in the same time slot should not pass the same intermediate nodes of the switch. The target of the scheduling is to transport all fluids with the minimum number of time slots.

To this end, we introduce a binary variable $y_{f,t}$ for all fluids $f \in \mathcal{F}$ and time slots $1 \leq t \leq |\mathcal{F}|$ to represent whether f is transported in t. We set the upper bound of t as $|\mathcal{F}|$, i.e. the number of to-be-transported fluids, to cover the worst-case that no concurrency exists. We introduce the following constraints to model that each fluid is transported in exactly one time slot:

$$\sum_{1 \leq t \leq |\mathcal{F}|} y_{f,t} = 1, \forall f \in \mathcal{F}. \tag{3}$$

Besides, we introduce a binary variable z_t for each time slot t to represent whether there is any fluid that will be transported in time slot t. We then introduce the following constraints:

$$z_t \geq y_{f,t}, \forall f \in \mathcal{F}, \forall 1 \leq t \leq |\mathcal{F}|. \tag{4}$$

If there is a fluid $f \in \mathcal{F}$ transported in time slot t, i.e $y_{f,t} = 1$, z_t will be forced to 1. Otherwise, z_t can be either 1 or 0. We then model the number of time slots needed for the fluid transportation as $\sum_{1 \leq t \leq |\mathcal{F}|} z_t$, and add it to the minimisation objective so that the solver will set z_t to 0 when no fluid is transported in time slot t.

In order to avoid the collision of different fluids transported in the same time slot, we introduce a binary variable $u_{f,n}$ for each fluid $f \in \mathcal{F}$ and each intermediate node $n \in \mathcal{N}$ to represent whether the transportation path of f includes node n. $u_{f,n}$ can thus be calculated with the following constraints:

$$u_{f,n} = \sum_{f^p \in \mathbb{P}} x_{f,f^p} \cdot b_{f^p,n}, \forall f \in \mathcal{F}, \forall n \in \mathcal{N}, \tag{5}$$

where x_{f,f^p} and $b_{f^p,n}$ are as introduced in Section III.B.

Next, we need to distinguish the multiplexing of fluids from the collision of fluids. Multiplexing refers to the case that after a fluid enters the switch, it branches at some intermediate nodes inside the switch and becomes multiple fluid flows following different paths. In this case, different branches are modelled as different fluids, which are transported in the same time slot but are allowed to pass the same intermediate nodes, as shown in Fig. 3(a). To prevent fluid collision without stopping fluid multiplexing, we divide the set $\mathcal{F}$ of fluids into disjoint subsets $F_1 \bigcup F_2 \bigcup \cdots \bigcup F_m = \mathcal{F}$ so that fluids branching from the same input pin are assigned to the same subset. We then introduce an integer variable $k_{i,n,t}$ to count the number of fluids in subset i that are transported in time slot t and pass node $n \in \mathcal{N}$. $k_{i,n,t}$ can thus be calculated with the following constraints:

$$k_{i,n,t} = \sum_{f \in \mathcal{F}_i} u_{f,n} \cdot y_{f,t}, \quad \forall 1 \leq i \leq m, \forall n \in \mathcal{N}, \forall 1 \leq t \leq |\mathcal{F}|. \tag{6}$$

Thus, we can prevent fluids in different subsets from passing the same intermediate node in the same time slot with the following constraints:

$$k_{i,n,t} \leq q_{i,n,t} \cdot m, \forall 1 \leq i \leq m, \forall n \in \mathcal{N}, \forall 1 \leq t \leq |\mathcal{F}|, \tag{7}$$

$$\sum_{1 \leq i \leq m} q_{i,n,t} \leq 1, \forall n \in \mathcal{N}, \forall 1 \leq t \leq |\mathcal{F}|, \tag{8}$$

where $q_{i,n,t}$ is an auxiliary binary variable representing whether node n is passed by any fluid in subset i in time slot t, and m is a constant equal to the largest index of the fluid subsets. Thus, if $k_{i,n,t}$ has positive value, $q_{i,n,t}$ will be set to 1 by constraint (7). Constraint (8) further rules that for any time slot t and

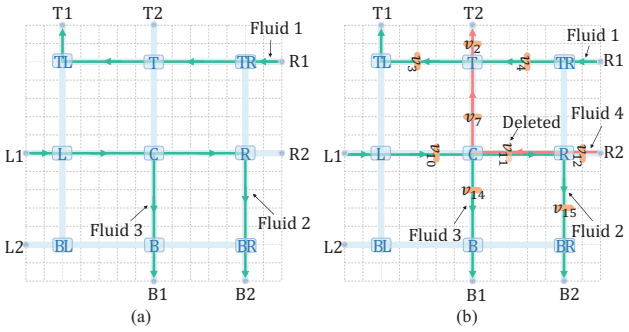

Fig. 3. (a) Possible transportation paths of three fluids in the same time slot. Fluids 2 and 3 branch from the same fluid that flows through pin L1 and are thus allowed to pass the same intermediate nodes, e.g. L and C. On the other hand, Fluid 1 cannot pass the same intermediate nodes as those passed by Fluids 2 or 3. (b) An 8-pin switch design to transport 4 fluids in two different time slots. Fluids transported in different time slots are denoted with different colours.

node n, at most one of the binary variables $q_{i,n,t}$ can be set to 1, which ensures that the node cannot be passed by fluids from different subsets in the same time slot.

D. Binding

To integrate our switch design into the mLSI synthesis flow, we need to specify the connection between the switch and other on-chip devices. To this end, we bind the pins of the switch to the pins of other on-chip devices. For convenience, we refer to pins of the switch as *switch pins* and the pins of other on-chip devices as *device pins*. We introduce a set $\mathcal{D}$ to contain all the device pins that need to be connected to the switch.

For each switch pin $p \in \mathcal{P}$ and each device pin $d \in \mathcal{D}$, we introduce a binary variable $v_{p,d}$ to represent whether p is bound to d. We then introduce the following constraints to model that each device pin must be connected to exactly one switch pin and each switch pin can be accessed by at most one device pin:

$$\sum_{p\in\mathcal{P}} v_{p,d} = 1, \forall d \in \mathcal{D}, \quad \sum_{d\in\mathcal{D}} v_{p,d} \leq 1, \forall p \in \mathcal{P}. \tag{9}$$

If a fluid $f \in \mathcal{F}$ is transported along path $f^p \in \mathbb{P}$, the switch pins p_s and p_e at the start and at the end of the path f^p must be bound to the source device pin d_s and the destination device pin d_e of the fluid, respectively. We model this requirement with the following constraints:

$$v_{p_s,d_s} \geq x_{f,f^p}, v_{p_e,d_e} \geq x_{f,f^p}, \forall f \in \mathcal{F}, \forall f^p \in \mathbb{P}. \tag{10}$$

To make our switch design compatible with different mLSI physical synthesis strategies, we provide three different *binding policies* for the users to specify the connection between other on-chip devices and the switch: *fixed*, *clockwise* and *unfixed*.

a) Fixed: Users directly define the value of the binding variables $v_{p,d}$ to specify a fixed switch pin $p \in \mathcal{P}$ for each device pin $d \in \mathcal{D}$. This policy can be applied when the physical synthesis tools integrate the switch after finishing the placement of all other on-chip devices and are thus confident about their binding decisions.

b) Clockwise: Users define an order of the device pins that require to be connected to the switch. The device pins are then bound clockwise to the switch pins. For example, if the user defines an order of three device pins $d_1 < d_2 < d_3$, a possible binding option is to bind d_1, d_2 and d_3 to switch pins on the top, right, and bottom boundary of the switch, respectively. In particular, the bounded switch pins do not need to be adjacent to each other, but their positions must follow the same direction as the hands on a clock. This policy can be applied when the physical synthesis tools have decided the rough positions of the on-chip devices before integrating the switch design. It gives more flexibility to the binding process than the fixed policy and is thus likely to achieve better fluid transportation performance. To implement this policy in our mathematical model, we index an arbitrary switch pin $p_1 \in \mathcal{P}$ as 1. Then starting from p_1, we visit all other switch pins in clockwise order and index them in ascending order. Thus, for device pins in the given order, the indices of the switch pins that they are bound to must form a sub-sequence of $1, 2, \cdots, |\mathcal{P}|, 1, 2, \cdots, |\mathcal{P}|$. To this end, we introduce an integer variable h_d for each device pin $d \in \mathcal{D}$ to represent the index of the switch pin p that d is bound to, i.e.:

$$h_d = \sum_{p\in\mathcal{P}} v_{p,d} \cdot i_p, \forall d \in \mathcal{D}, \tag{11}$$

where i_p is the index of the switch pin p. Then we can achieve the clockwise binding solutions with the following constraints:

$$h_d \leq h_{d+1} - 1 + q_d \cdot |\mathcal{P}|, \forall d \in \mathcal{D}, \quad \sum_{d\in\mathcal{D}} q_d = 1, \tag{12}$$

where q_d is an auxiliary binary variable representing whether h_d is the largest index among all bound switch pins, and $d+1$ is the successive device pin of d in the given order. If d is the last pin in the order, $d+1$ will be the first pin in the order. If $q_d = 0$, h_d must be smaller than h_{d+1}; and if $q_d = 1$, the inequation becomes a tautology and thus does not confine the relation between h_d and h_{d+1}, which implies the case that d is bound to the switch pin with the largest index among all bound switch pins, e.g. $h_d = |\mathcal{P}|$ and $h_{d+1} = 1$.

c) Unfixed: Users do not define any binding specifications. This policy can be applied when the switch is the key component of the mLSI design and thus other on-chip devices should be synthesised based on the binding results, e.g. in mLSI designs for sample preparation applications. The unfixed policy has the most flexibility and can theoretically achieves the best fluid transportation performance. For this policy, we do not need additional binding constraints beyond constraints (9)-(10).

E. Objective

Our objective is to synthesise a switch that transports all fluids with least time and fewest resources. We model the number of time slots required for the transportation as $\sum_{1\leq t\leq\mathcal{F}} z_t$, as introduced in Section III.C. Besides, we model the resources required for the transportation as the total length of the flow channel segments in the switch. To this end, we introduce a binary variable w_s for each segment $s \in \mathcal{S}$ to represent whether s is required for the transportation. We then introduce the following constraint to model that a segment s is required, if a path f^p containing s is used to transport any fluid:

$$w_s \cdot |\mathcal{F}| \geq \sum_{f\in\mathcal{F}} x_{f,f^p}, \forall s \in \mathcal{S}, \forall f^p \in \mathbb{P} \wedge f^p \text{ containing } s \tag{13}$$

Thus, the objective function of our model is:

$$Minimise : \alpha \sum_{1\leq t\leq|\mathcal{F}|} z_t + \beta \sum_{s\in\mathcal{S}} w_s \cdot l_s. \tag{14}$$

where α and β are adjustable weight coefficients, and l_s is a constant representing the length of the flow channel segment s.

F. Valve Reduction and Pressure Sharing

Valves are essential mLSI-components to guide fluid directions. To actuate a valve, we need to connect the valve via control channels to a pressure inlet. Since an inlet usually occupies much larger chip area than other on-chip components, it is preferable to avoid redundant implementation of valves and to make multiple valves share the pressure provided by the same inlet. In this work, we propose a method to identify redundant

valves, and to allow other valves to share pressure so that the number of required inlets is minimised.

Our switch design initially has a placeholder for valve on every flow channel segment $s \in \mathcal{S}$. However, we only need the valve on s, when we need to temporarily block s to prevent fluids in an adjacent segment of s from entering s. On the other hand, if s is in the transportation path of all the fluids passing through its adjacent segments, the valve on s can be removed. Thus, given a switch design, for each segment s in the design, we build a set $\mathcal{F}_s \subseteq \mathcal{F}$ that contains all the fluids that need to be transported by s, and a set $\mathcal{A}_s \subseteq \mathcal{S}$ that contains all the adjacent segments of s. We can derive that the valve on s is redundant, if $\forall a \in \mathcal{A}_s$, $\mathcal{F}_a \subseteq \mathcal{F}_s$.

For example, Fig. 3(b) shows an 8-pin switch design for the transportation of four fluids. We denote the fluids as f_1, f_2, f_3 and f_4, respectively. In particular, f_2 and f_3 branch from the same origin by multiplexing and can thus be considered as identical. For segment (C, R) in the design, $\mathcal{F}_{(C,R)} = \{f_2, f_4\}$, and $\mathcal{A}_{(C,R)} = \{(L, C), (C, T), (C, B), (R, BR), (R, R2)\}$. Further, $\mathcal{F}_{(L,C)} = \{f_2\} \subseteq \mathcal{F}_{(C,R)}$, $\mathcal{F}_{(C,T)} = \{f_4\} \subseteq \mathcal{F}_{(C,R)}$, $\mathcal{F}_{(C,B)} = \{f_3\} = \{f_2\} \subseteq \mathcal{F}_{(C,R)}$, $\mathcal{F}_{(R,BR)} = \{f_2\} \subseteq \mathcal{F}_{(C,R)}$, and $\mathcal{F}_{(R,R2)} = \{f_4\} \subseteq \mathcal{F}_{(C,R)}$. Thus, the valve on (C, R) is redundant and can be deleted from the design.

After removing the redundant valves, we aim to control the rest of the valves with the fewest inlets. To this end, we first identify valves that can potentially share the same pressure. This can be done by analysing the valve actuation status in different time slots [2], [8]. Specifically, we can denote the actuation status of a valve v in a time slot t as *open*, *closed*, or *irrelevant*, among which *open* and *closed* conflict with each other, and *irrelevant* is compatible with any status. If there is no conflicting actuation status between two different valves in all time slots, the two valves can potentially share the same pressure.

We then build a graph model $G = (V, E)$ to represent the potential pressure sharing options, in which V is the set of vertices representing valves, and E is the set of edges representing a pair of valves that can share pressure. Thus, a clique in G, i.e. a subset $V' \subseteq V$ such that every two vertices in V' are adjacent, represents a set of valves that can share the same pressure. In this manner, we transform the pressure sharing problem into a minimum clique cover problem, i.e. to find the minimum number of inlets required to support the fluid transportation, we just need to find the minimum number of cliques to cover all the vertices in the graph. Since minimum clique cover problems are in general NP-hard, we solve this problem with integer linear programming (ILP). As minimum clique cover is a well-researched graph problem, we omit the the detailed variables and constraints for the ILP model.

IV. Experimental Results

We implemented the proposed methods with C++ and solved the mathematical models employing the Gurobi Optimizer [9] on a computer with an Intel Core 2 Duo E8400 3.00 GHz CPU. All test cases used for our experiments are from real-world bio-applications [10]–[13]. In all the experiments, we fix the weight coefficients α as 1 and β as 100.

TABLE I
Synthesis Results

Application	# d	type	T(s)	L(mm)	#v	#t	#i
ChIP-1 [10]	9	12-pin	10865	13.6	6	2	2
nucleic acid p. [11]	7	8-pin	3	9.8	6	2	2
mRNA iso. [12]	10	12-pin	6665	17.8	8	2	2

$\#d$: number of device pins; T: run time; L: flow channel length; $\#v$: number of valves; $\#t$: number of time slots; $\#i$: number of pressure inlets.

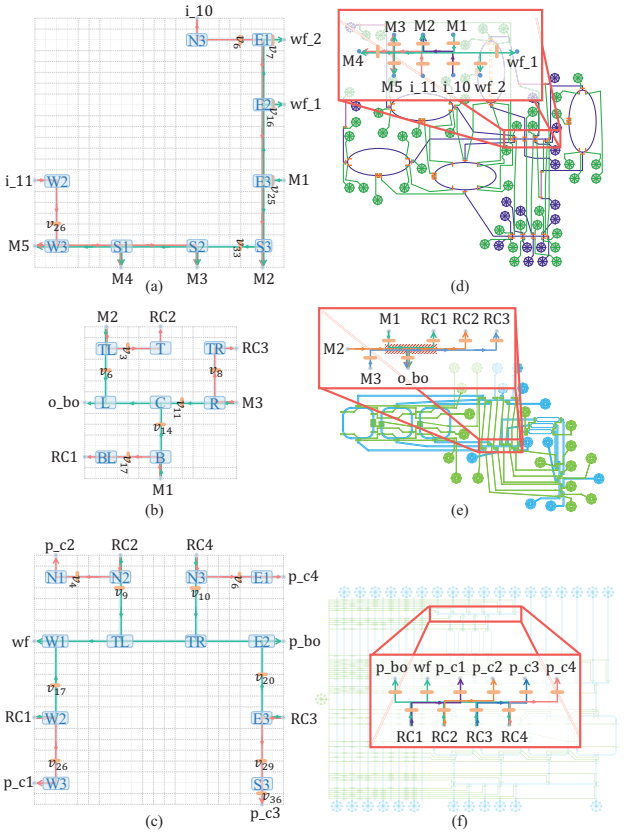

Fig. 4. Comparison between the switch synthesised by our method and the switch synthesised by Columba for three different applications: (a)(d) ChIP; (b)(e) nucleic acid processor; (c)(f) mRNA isolation.

A. Comparison with conventional switch designs

We first synthesise three switch designs applying the unfixed policy for different bio-applications with conflicting fluids. We compare our switches with the switches synthesised by the state-of-the-art physical design tool Columba for the same test cases using the cloud platform of Columba [14]. Table I shows the synthesis results of this work. Fig. 4 demonstrates the fluid transportation paths in our switch design and shows the comparison between the switches of this work and by Columba.

In general, for all three applications, our switches successfully support contamination-free transportation paths for all the fluids, while the Columba switches fail to prevent fluid contamination for the first two applications. Specifically, in the ChIP application, there are two conflicting fluids from fluid

TABLE II
SYNTHESIS RESULTS APPLYING DIFFERENT BINDING POLICIES

Application	$\#d$	type	Policy	$T\,(s)$	$L\,(mm)$
ChIP-2 [10]	10	12-Pin	Fixed	0.2	180
			Clockwise	30.4	154
			Unfixed	3763.3	154
Kinase Activity-1 [13]	4	8-Pin	Fixed	0.03	46
			Clockwise	0.68	46
			Unfixed	1.49	46
Kinase Activity-2 [13]	5	8-Pin	Fixed	0.04	60
			Clockwise	2.1	60
			Unfixed	8.4	60

inlet i_{10} to mixer $M2$ and from fluid inlet i_{11} to mixer $M3$, $M4$, and $M5$, respectively. In particular, the latter requires fluid multiplexing. The transportation paths of the two fluids in our switch design do not consist of overlapping nodes and thus contamination is avoided, while the Columba switch routes all fluids via the same spine and thus cannot avoid contamination, as shown in Fig. 4(a) and (d), respectively. In the nucleic acid processor application, the products of mixing operations are conflicting fluids that need to be transported from each different mixer to its corresponding chamber. Our switch avoids the overlapping of all fluid transportation paths, while the Columba switch suffers heavy contamination in the spine, as shown in Fig. 4(b) and (e), respectively. We mark the segment that is contaminated the most with red stripes, as every fluid needs to pass through it. In the mRNA isolation application, conflicting fluids need to be transported from $RC1$, $RC2$, $RC3$ and $RC4$ to fluid outlets p_c1, p_c2, p_c3 and p_c4, respectively. As shown in Fig. 4(c) and (f), both our switch and the Columba switch avoid the contamination, but the Columba switch cannot support concurrent transportation of multiple fluids and thus cannot transport the fluids as efficiently as our switch.

We can also notice that compared to the switches synthesised by Columba, our switches have longer flow channels. However, the additional channel segments not only allow us to avoid fluid contamination, but also allow us to transport multiple fluids concurrently: for all three applications, we were able to transport all fluids in only two time slots. Besides, thanks to our valve reduction and pressure sharing methods, we transport the fluids with much fewer valves and pressure inlets. For example, our switch for the ChIP application only require 6 valves and 2 pressure inlets, while the Columba switch requires 9 valves and 9 pressure inlets.

B. Comparison of different Binding Policies

We further synthesise switch designs with different binding policies, i.e. *fixed*, *clockwise*, and *unfixed*, for three applications without conflicting fluids. For the fixed and the clockwise policies, we define the binding variables and the binding orders based on the position of the on-chip devices in the designs synthesised by Columba. Table II shows the synthesis results.

In general, the fixed policy requires the least program run time to synthesise the switch, but due to the confinement of the solution space, it cannot always find the best fluid transportation option. The clockwise policy also requires little time for the synthesis. Besides, the synthesised switches support comparable fluid transportation performance as the unfixed policy. The unfixed policy has the largest solution space and can thus always achieve the best performance among all three policies. However, it may require more time for the synthesis. To note is that most of the program run time is spent on proving the optimality of the solutions but not on finding the solutions.

V. CONCLUSION

In this work, we propose a mathematical method to synthesise contamination-free switch designs for mLSI applications. Compared to the state-of-the-art methods, our method prevents fluid contamination without prolonging the assay execution time, enables concurrent transportation of multiple fluids to improve the throughput, and reduces the number of valves and pressure inlets required for the transportation.

REFERENCES

[1] T.-M. Tseng, M. Li, B. Li, T.-Y. Ho, and U. Schlichtmann, "Columba: co-layout synthesis for continuous-flow microfluidic biochips," in *ACM/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[2] T.-M. Tseng, M. Li, D. N. Freitas, T. McAuley, B. Li, T.-Y. Ho, I. E. Araci, and U. Schlichtmann, "Columba 2.0: A co-layout synthesis tool for continuous-flow microfluidic biochips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1588–1601, 2018.

[3] T.-M. Tseng, M. Li, D. N. Freitas, A. Mongersun, I. E. Araci, T.-Y. Ho, and U. Schlichtmann, "Columba S: A scalable co-layout design automation tool for microfluidic large-scale integration," in *ACM/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[4] K. Hu, K. Chakrabarty, and T.-Y. Ho, "Wash optimization for cross-contamination removal," in *Computer-Aided Design of Microfluidic Very Large Scale Integration (mVLSI) Biochips*. Springer, 2017, pp. 53–79.

[5] X. Huang, W. Guo, Z. Chen, B. Li, T.-Y. Ho, and U. Schlichtmann, "Flow-based microfluidic biochips with distributed channel storage: Synthesis, physical design, and wash optimization," *IEEE Transactions on Computers*, 2021.

[6] K. Dorfman, M. Chabert, J.-H. Codarbox, G. Rousseau, P. Cremoux, and J.-L. Viovy, "Contamination-free continuous flow microfluidic polymerase chain reaction for quantitative and clinical applications," *Analytical chemistry*, vol. 77, pp. 3700–3704, 2005.

[7] M. Li, T.-M. Tseng, B. Li, T.-Y. Ho, and U. Schlichtmann, "Sieve-valve-aware synthesis of flow-based microfluidic biochips considering specific biological execution limitations," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 624–629.

[8] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer routing and control-pin minimization for flow-based microfluidic biochips," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 55–68, 2016.

[9] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," http://www.gurobi.com.

[10] A. Wu, J. Hiatt, R. Lu, J. Attema, N. Lobo, I. Weissman, M. Clarke, and S. Quake, "Automated microfluidic chromatin immunoprecipitation from 2,000 cells," *Lab on a chip*, vol. 9, pp. 1365–1370, 2009.

[11] D.-W. Cho, V. Studer, G. Hang, W. Anderson, and S. Quake, "A nanoliter-scale nucleic acid processor with parallel architecture," *Nature Biotechnology*, vol. 22, pp. 435–439, 2004.

[12] J. Marcus, W. Anderson, and S. Quake, "Microfluidic single-cell mrna isolation and analysis," *Analytical chemistry*, vol. 78, pp. 3084–3089, 2006.

[13] C. Fang, Y. Wang, N. Vu, W.-Y. Lin, Y.-T. Hsieh, L. Rubbi, M. Phelps, M. Müschen, Y.-M. Kim, A. Chatziioannou, H.-R. Tseng, and T. Graeber, "Integrated microfluidic and imaging platform for a kinase activity radioassay to analyze minute patient cancer samples," *Cancer research*, vol. 70, pp. 8299–8308, 11 2010.

[14] T.-M. Tseng, M. Li, Y. Zhang, T. Y. Ho, and U. Schlichtmann, "Cloud columba: Accessible design automation platform for production and inspiration," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.