Multi-Channel and Fault-Tolerant Control Multiplexing for Flow-Based Microfluidic Biochips

Ying Zhu[1], Bing Li[1], Tsung-Yi Ho[2,5], Qin Wang[3], Hailong Yao[3], Robert Wille[4], Ulf Schlichtmann[1]
[1]Chair of Electronic Design Automation, Technical University of Munich, Germany [2]National Tsing Hua University, Hsinchu, Taiwan
[3]Tsinghua University, Beijing, China [4]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[5]Institute for Advanced Study, Technical University of Munich, Germany
{ying.zhu, b.li, ulf.schlichtmann}@tum.de, {ho.tsungyi, qinwangthu}@gmail.com, hailongyao@mail.tsinghua.edu.cn, robert.wille@jku.at

ABSTRACT

Continuous flow-based biochips are one of the promising platforms used in biochemical and pharmaceutical laboratories due to their efficiency and low costs. Inside such a chip, fluid volumes of nanoliter size are transported between devices for various operations, such as mixing and detection. The transportation channels and corresponding operation devices are controlled by microvalves driven by external pressure sources. Since assigning an independent pressure source to every microvalve would be impractical due to high costs and limited system dimensions, states of microvalves are switched using a control logic by time multiplexing. Existing control logic designs, however, still switch only a single control channel per operation — leading to a low efficiency. In this paper, we propose the first automatic synthesis approach for a control logic that is able to switch multiple control channels simultaneously to reduce the overall switching time of valve states. In addition, we propose the first fault-aware design in control logic to introduce redundant control paths to maintain the correct function even when manufacturing defects occur. Compared with the existing direct connection method, the proposed multi-channel switching mechanism can reduce the switching time of valve states by up to 64%. In addition, all control paths for fault tolerance have been realized.

1 INTRODUCTION

In traditional biochemical laboratories, experiments are performed using cumbersome devices such as tubes and droppers. This work flow is inconvenient and error-prone, due to human intervention required for the experiment process. Even in the more modern system-in-a-package experiment systems, only relatively simple experiment protocols can be processed automatically, and complex biochemical experiments such as exhaustive diagnosis of diseases still cannot completely avoid human intervention.

To overcome the shortcomings of the experiment systems above, flow-based microfluidic biochips have been investigated intensely in the past decade [1]. In such a chip, a large number of devices, e.g., mixers and detectors, are constructed. These devices are connected by microchannels to transport intermediate experiment results. The transportation of these results is controlled by microvalves which are tiny switches built on top of the channels. The open/closed states of these valves are controlled by air pressure patterns generated by a control logic. The control logic is managed by a microcontroller so that the execution of a biochemical protocol can completely be automated.

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

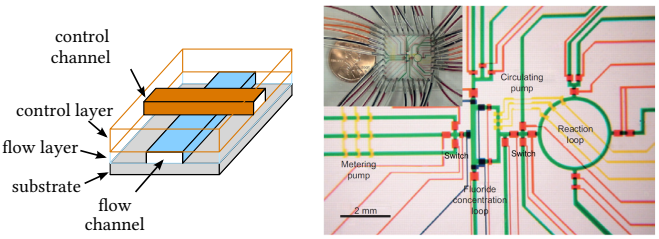
ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240830

A major advantage of biochips is their large integration. Accordingly, the manufacturing process of biochips has taken a road similar to integrated circuits by etching flow channels and control channels on a substrate [2]. Observing this similarity, the design automation community has started to propose methods and work flows to improve the design quality and efficiency. For example, the synthesis of biochip architectures has been addressed in [3–5] and the routing of flow channels in [6, 7]. Furthermore, test methods using an ATPG-based method for defect detection in biochips after manufacturing has also been proposed in [8].

Compared to integrated circuits, biochips, however, still have their own specific features. Besides flow channels which are used to transport fluid samples between devices, valves need to be driven by external air pressure patterns to change their states according to the protocols of applications. The structure of a valve is shown in Fig. 1(a), where the control channel is built on top of the flow channel. An air pressure through the control channel squeezes the flow channel to block the movement of flow samples. When the air pressure in the control channel is released, the flow channel opens again for fluid transportation. In other words, a valve works like a switch, whose state is controlled by the air pressure in the control channel. With valves as the controlling units, complex biochips can be constructed, as shown in Fig. 1(b).

When executing an application, the patterns of air pressure in the control channels should be generated by a control logic, which plays a critical role in a biochip, since it manages the overall execution of applications. Recently, related research considering control channel optimization has started to appear. For example, the method in [9] minimizes pressure-propagation delay in control channels to reduce the response time of valves. The lengths of control channels are matched in [10] to synchronize switching times of valves. These methods, however, mainly focus on the control channels that deliver air pressure to valves. The control logic to generate the required pressure patterns, however, has not sufficiently been investigated yet. Up to now, only one method has been proposed to consider the reliability of control logic [11], where the order of patterns that are required to control valves is adjusted to reduce the maximum number of switching activities in the control logic. This method, unfortunately, still does not address the efficiency of generating the required pressure patterns.

In this paper, we examine the design of control logic and propose a method to improve its efficiency in generating the required pressure patterns. In addition, the resources required by the control logic are also reduced. Our contributions are twofold. First, the basic design rules of control logic are examined and a new concept of switching *multiple* control channels simultaneously by expressing channel switching patterns with Boolean logic is proposed for the first time. With this concept, the efficiency of generating the required pressure patterns can be improved by up to 64%. Second, the structure of the control logic is determined by mapping the identified control patterns onto a general routing grid. Since this mapping allows control channels to be routed horizontally and vertically, it provides more flexibility in determining



(a) (b)

Figure 1: (a)Valve structure. (b) A flow-based biochip with flow channels (green) and control channels (yellow and red) [12] (b).

new structures of control logic. To the best of our knowledge, this is the first work to examine the design of control logic itself.

The rest of this paper is organized as follows. In Section 2, the existing structure of control logic design is explained. In Section 3, the basic ideas of the proposed multi-channel switching and fault-tolerance are described. The implementation of these ideas on a general routing grid is explained in Section 4. Simulation results are reported in Section 5. Conclusions are drawn in Section 6.

2 CONTROL LOGIC IN FLOW-BASED BIOCHIPS

In flow-based biochips, valves at the intersections of flow and control channels need to be switched by the patterns of air pressure generated by the control logic.

In Fig. 2 an example of a complete biochip from [13] is shown. In Fig. 2(a), the flow core of the biochip is located in the center, which is enlarged in Fig. 2(b). This flow core is designed as a valve array for its full reconfigurability. Each block in Fig. 2(b) is a valve with a similar structure shown in Fig. 1(a) and the thin lines in Fig. 2(b) are control channels to conduct air pressure to the valves. Compared to the biochip in Fig. 1(b), this regular valve array can execute any transportation and mixing operations by dynamic mapping [14] and can easily be tested after manufacturing [15].

In controlling the valves inside the flow core, it is not practical to assign each valve an independent pressure source, due to the cost and the size of these mechanical devices. For example, in the design in Fig. 2(a), 114 valves in the flow core have been implemented. For executing applications, instead of using 114 pressure sources directly, which would be very cumbersome and expensive, only 15 pressure sources are used to delivery pressure patterns, including 14 control ports and one core input.

The reduction of the number of pressure sources is implemented by a multiplexing system, called control logic henceforth as shown in Fig. 2(a), where the control logic and the control channels take the major area of the biochip. At the bottom of Fig. 2(a), the core input provides a pressure source that can be switched on or off. On the right, the control ports create control patterns that specify which control channel can be connected to the core input to update its pressure state. The control logic in the middle forms a multiplexing function to connect the channels to the core input according to these control patterns. Once a control channel is connected to the core input, its pressure state is updated to the same as that of the core input. Correspondingly, the open/closed state of the valve in the flow core driven by this control channel is also updated. In the following, the valves in the flow core are called ***flow valves*** and they share the same indices as the control channels.

The multiplexing function of the control logic to reduce the number of pressure sources is explained using Fig. 3(a). In this example, four external ports x_1, $\overline{x}_1$, x_2, $\overline{x}_2$ are connected to pressure sources to control the connection of the control channel that drive the three valves in the flow core. In the following, these external ports are called

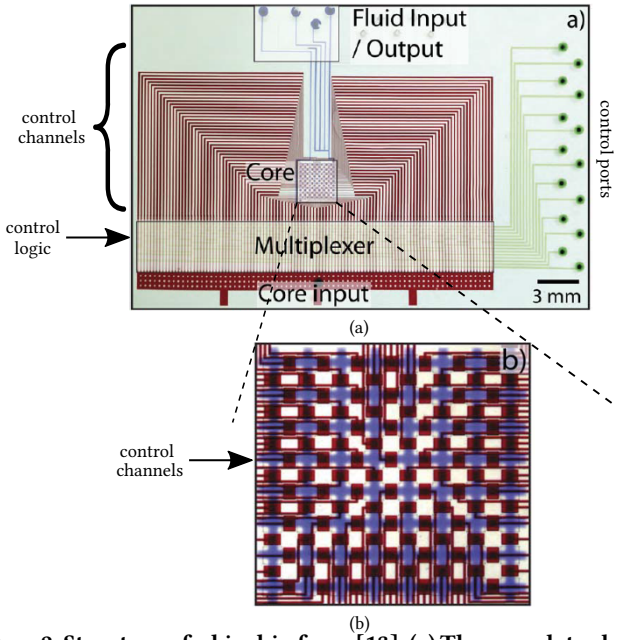


Figure 2: Structure of a biochip from [13]. (a) The complete chip with flow part and the control logic. (b) Enlarged flow part.

control ports. In control logic design, the pressure values of the control ports are often complementary [13, 16]. At any time, only one of a pair of complementary ports can have a high pressure, so that the complementary control variables x_i and $\overline{x}_i$ can be implemented. These variables are used to control the valves built on top of the channels in the control logic, called ***control valves*** as shown in Fig. 3. The channels inside the control logic are called ***control paths***. The outputs of the control logic represent the states of the control channels, and are called ***control outputs***. The states of control ports and the control valves determine which control path is to be connected to the core input to change the valves of the control outputs. In the following, the states of control outputs and the states of control channels will be used interchangeably. For example, control channel 1 driving flow valve 1 is connected to control output 1, whose value is updated to the value of the core input when both x_1 and x_2 are set to logic '1'.

The combinations of control valves on the control paths form ***control patterns*** for channel multiplexing. For example, three control patterns x_1x_2, $\overline{x}_1x_2$, and $x_1\overline{x}_2$ are used in Fig. 3 to control the three channels. At any moment, only one of these patterns can be true, so that only one control output can be connected to the core input for updating its pressure state. If the target pressure in the control channel should be high, the pressure of the core input is activated; otherwise, the core input releases the pressure in the control channel. With this mechanism, n control ports can be used to multiplex $2^{n/2}$ control channels. If the number of control channels is between $2^{n/2-1}$ and $2^{n/2}$, some control patterns are not used, such as $\overline{x}_1\overline{x}_2$ in Fig. 3. In the control logic in Fig. 3, the number of pressure sources is five, which is larger than the number of control channels. Therefore, the control multiplexing actually requires more pressures sources in this case. However, as the number of control channels n increases, the required number of pressure sources $2 * \lceil \log_2 n \rceil + 1$ rapidly decreases compared to n.

The function of the control logic shown in Fig. 3 is to change the pressure values in the control channels so that flow valves can be switched to execute applications. These pressure values are called ***channel states***. Assume that at time t the channel states are "011", where '1' represents that the pressure in the corresponding control

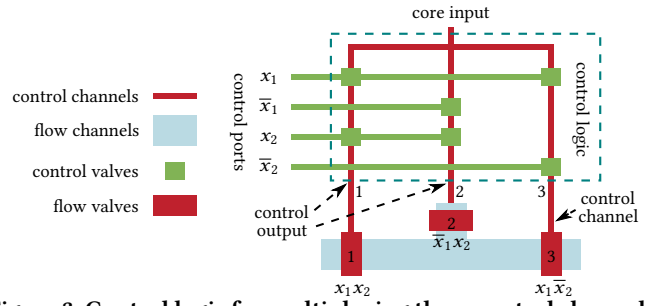


Figure 3: Control logic for multiplexing three control channels. Control ports x_1, $\overline{x}_1$, x_2, and $\overline{x}_2$ are connected to external pressure sources.

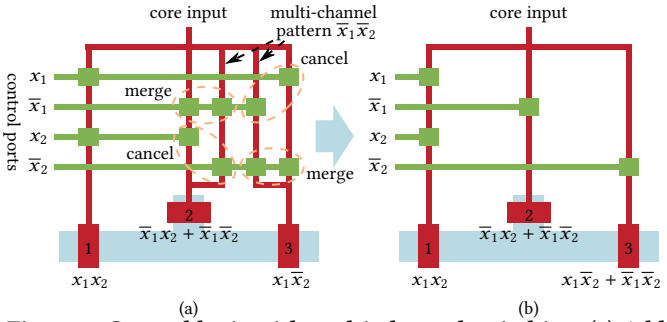


Figure 4: Control logic with multi-channel switching. (a) Additional control pattern $\overline{x}_1\overline{x}_2$ is used to update control channels 2 and 3 simultaneously. (b) Simplified control logic after valve merging and canceling.

channel is high and '0' represents the pressure is low. At the time $t+1$, assume that the states of the control channels need to be updated to "100". Since the Boolean control logic in Fig. 3 only allows one control channel to be opened at a moment, the state transitions need to be implemented using three switching operations, in which the control variables x_1 and x_2 are set to "11", "10" and "01", respectively. In this process, the three control channels are connected to the core input one after the other, activated by the control patterns x_1x_2, $\overline{x}_1x_2$, and $x_1\overline{x}_2$, respectively. Accordingly, the pressure of the core input should be set to '1', '0' and '0' to update the pressures in the control channels. For convenience, the period of the states of all control channels at time t to new states at time $t+1$ is called a ***time slot***. Within a time slot, the states of several control channels may need to be changed by the control logic. Therefore, the state transition from time slot t to time slot $t+1$ may be split into several ***time slices***, each of which represents an actuation of the control logic.

3 CONCEPTS OF MULTI-CHANNEL SWITCHING AND FAULT-TOLERANCE IN CONTROL LOGIC

The control logic design described above is very effective in reducing the number of pressure sources. However, flow valves are switched one after another in this scheme by activating control channels individually. During the state transition from time slot t to time slot $t+1$, the execution of an application on the biochip is paused. If the number of valves whose states should be updated is large, the execution time of the application can be prolonged. This disadvantage above is due to the fact that only one output can be updated at a moment. To solve this problem, a new design scheme that allows multiple control paths to be activated simultaneously will be introduced in the following to improve the efficiency of control logic.

In addition, the existing control logic design is also sensitive to manufacturing defects. If a control channel cannot be opened properly, the corresponding flow valve cannot be switched anymore, potentially leading to a complete chip failure. This reliability issue is addressed in the proposed new design scheme with duplicated control paths, which are constructed together with control paths for multiple-channel switching to improve design efficiency.

3.1 Multi-channel switching

In Fig. 3, only three flow valves are driven by the control logic, although the combinations of pressure sources are capable of generating four control patterns. Consider the scenario that channel states are switched from "011"→"100". The control logic individually switches the second and the third channel from '1' to '0'. Therefore, it is possible to combine the last two operations. Besides the three control patterns used in Fig. 3, there is still the fourth control pattern $\overline{x}_1\overline{x}_2$ available, which can be used to switch the channel 2 and 3 together, as shown in Fig. 4(a). In this augmented design, both channels 2 and 3 are connected to the core input through the newly added control paths which are opened by the pattern $\overline{x}_1\overline{x}_2$. Consequently, in the transition from "011"→"100", the number of time slices can be reduced by 1.

In Fig. 4(a), flow valve 3 is driven by two control paths. At the bottom of these two paths, the two control valves are connected to the same control port $\overline{x}_2$. Therefore, they can be merged to save one valve. The two control valves at the top of these two control paths are complementary, since they are connected to x_1 and $\overline{x}_1$. Therefore, no matter what value x_1 has, at least one of the two control paths to flow valve 3 opens on the condition that $\overline{x}_2$ is set to '1'. Accordingly, the two valves at the top of the two control paths to flow valve 3 can be canceled. The merging and canceling operations can also be applied to the control channels to flow valve 2 in the control logic. Consequently, the control logic can be simplified as shown in Fig. 4(b), where only one control valve is required in each of the control paths to the control outputs 2 and 3. This merging and canceling process is actually the simplification of the Boolean logic $\overline{x}_1x_2 + \overline{x}_1\overline{x}_2 = \overline{x}_1$ and $x_1\overline{x}_2 + \overline{x}_1\overline{x}_2 = \overline{x}_2$. The + sign means that either control path can drive the corresponding flow valve sufficiently. In Fig. 4(b) the number of valves has been reduced from 10 to 4 compared to Fig. 4(a). Compared to the original control logic in Fig. 3, the number of valves has also been reduced from 6 to 4, while the multi-channel switching function is still implemented.

In the simplified design in Fig. 4(b), the flow valves can still be switched individually, because the individual control patterns $\overline{x}_1x_2$ and $x_1\overline{x}_2$ are still valid for channels 2 and 3 respectively. For example, the control pattern $x_1\overline{x}_2$ connects only the control channel 3 to the core input, while the other two channels are still closed. Consider a more complex scenario of channel states "011"→"100" →"001"→"110". The transition "100" →"001" requires two time slices for channels 1 and 3, while channel 2 does not need to be updated. The transition "001"→"110" still requires three time slices, since the channels 1 and 2 cannot be updated simultaneously. Consequently, the number of total time slices required by the flow valves can be calculated as the sum of time slices in the time slots, i.e., 2+2+3=7, which is less than the time slices 8 required in the original design in Fig. 3, where only single-channel switching is possible.

Besides the logic simplification with respect to a single flow valve, control valves can also be merged with respect to the core input. For example, the three control valves connected to $\overline{x}_1$ in Fig. 4(a) can also be merged, because these valves always open or close simultaneously to allow or block the pressure from the core input to the next segments of the control paths. This merging operation requires that the logic expressions from the core input to several neighboring control valves are equal, leading to logic simplification with respect to internal nodes inside the control logic. In designing the control logic, this type of

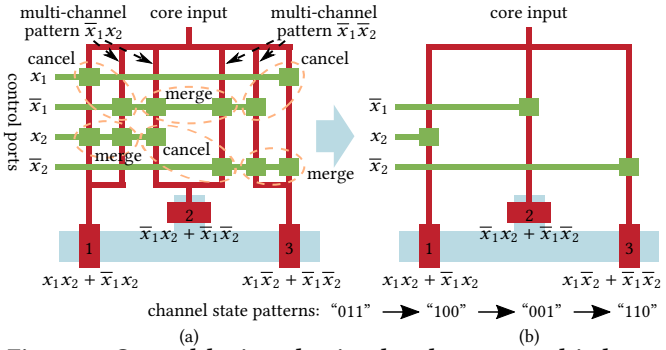


Figure 5: Control logic reduction by alternate multi-channel switching. (a) Control pattern $\overline{x}_1x_2$ is used to update control channels 1 and 2 simultaneously and control channel 2 has no individual control pattern. (b) Simplified control logic.

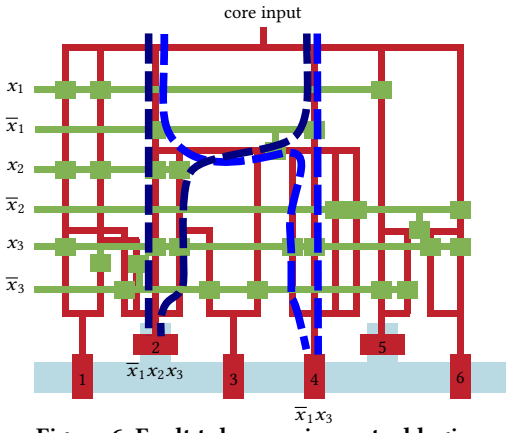


Figure 6: Fault tolerance in control logic.

simplification and that with respect to individual flow valves need to be balanced. For example, it is more beneficial to cancel the valves in Fig. 4(a) first than merging the three valves connected to $\overline{x}_1$.

3.2 Logic reduction by alternate multi-channel switching for given applications

In the case in Fig. 4(b), the control logic cannot be reduced anymore, since all the spare control patterns have been used. This design still maintains the ability to update each control channel individually, as well as to update the states of the channels 2 and 3 simultaneously. The maintained single-switching ability guarantees that this control logic is capable of generating states of control channels for any applications.

If the application of the biochip is given, the state transitions become known. In a sequence of transitions such as "011"→"100"→"001"→"110", it can be observed that the control channel in middle is always updated together with another one, either the first or the last. This phenomenon indicates that it is not necessary to assign channel 2 an individual control pattern. Instead, the original control pattern $\overline{x}_1x_2$ in Fig. 4(a) can be spared to implement multi-channel switching between channels 1 and 2, as shown in Fig. 5(a).

In Fig. 5(a), control channels 1 and 3 receive individual control patterns x_1x_2 and $x_1\overline{x}_2$, respectively. The control channel 2, however, can only be switched together with either channel 1 by $\overline{x}_1x_2$ or channel 3 by $\overline{x}_1\overline{x}_2$. This loss of generality makes this control logic design depend on a given application. But the control logic itself can be simplified and the switching time of valves in executing the application can be reduced.

After the merging and canceling operations are applied to the case in Fig. 5(a), only three control valves are left in the design, as shown in Fig. 5(b). The logic of the control patterns can be verified from the multi-channel control patterns as $x_1x_2 + \overline{x}_1x_2 = x_2$ for channel 1, $\overline{x}_1x_2 + \overline{x}_1\overline{x}_2 = \overline{x}_1$ for channel 2, and $x_1\overline{x}_2 + \overline{x}_1\overline{x}_2 = \overline{x}_2$ for channel 3. Furthermore, the number of control ports is also reduced by one, since x_1 is not required anymore, leading to a further decrease of the complexity of the biochip platform.

For the state transitions of the control channels "011"→"100"→"001"→"110", the further simplified control logic requires only 2+2+2=6 time slices, since channel 2 always shares the new value with another channel. A special case is in the transition "100"→"001", where the state of channel 2 does not change. Therefore, the function is regardless of whether it is updated or not, similar to a don't care in logic design. In the design in Fig. 5(b), the pattern $\overline{x}_1x_2$ takes advantage of this phenomenon for multi-channel switching. Since the number of control valves in the control logic has also been reduced significantly, this comparison confirms that the newly introduced multi-channel switching concept can improve the execution efficiency of the control logic and reduce the resource usage at the same time.

The example in Fig. 5(b) is not very complicated to design directly, since there are four variables x_1, $\overline{x}_1$, x_2, and $\overline{x}_2$ available for control patterns of only three channels. In fact, any three of the four variables x_1, $\overline{x}_1$, x_2, and $\overline{x}_2$ can be used to form a solution of the control logic with multi-channel switching shown in Fig. 5(b). However, in reality the number of control channels is usually larger than the number of control variables, so that a feasible solution with multi-channel switching becomes not straightforward anymore. In Section 4, the assignment of control patterns and the construction of control logic will be solved by integer linear programming (ILP) formulation.

3.3 Fault tolerance in control logic

In Fig. 5(b), there is only one valve and one control path to a control output. During manufacturing, there might be defects in the control logic. If a control valve cannot be closed, the core input is always connected to the control channel, leading to a failed flow valve in the biochip. To tackle this problem, a control valve can be duplicated and inserted in series to the original control valve, similar to the solution in [17, 18]. On the other hand, if a control valve cannot be opened or a control path is blocked, there is no path to connect the core input to the control output to update its state. A simple strategy to solve this problem is to duplicate all the channels and valves and insert them in parallel to the original channels and valves. This method, however, may lead to an unnecessarily complicated design and large resource usage.

Figure 6 shows another example of control logic generated by the proposed method, where the control paths along control valves to control outputs 2 and 4 are highlighted. In this case, the control pattern $\overline{x}_1x_2x_3$ activates these two outputs simultaneously, forming a multi-channel switching pair. Furthermore, to each of these control outputs, there are two independent paths through the control logic. If one of these paths is blocked due to a manufacturing defect, the other path still maintains the correct function of the control logic.

Compared with the straightforward strategy to duplicate the control logic to provide fault tolerance, the control paths in Fig. 6 share control valves, e.g., the two valves connected to $\overline{x}_1$, leading to a reduction of resource usage. To design such a control logic with efficient multi-channel switching and resource sharing for fault-tolerance, these features need to be considered together in a general framework.

4 A GENERAL FRAMEWORK FOR CONTROL MULTIPLEXING AND FAULT TOLERANCE

To generate a control logic supporting multi-channel switching and fault tolerance, two major steps are used in the proposed method. First, the given channel states are converted to channel switching patterns. Then control channels that should be enabled simultaneously are identified to reduce the total number of time slices. In the second step, the control logic is constructed on a virtual grid to meet the multi-channel switching and fault tolerance requirements. In addition, the total number of control valves in the control logic is also minimized. Both steps are formulated into integer programming problems (ILP) in the following and solved by a solver in the proposed method.

4.1 Determining multi-channel switching scheme from switching patterns

As discussed in Section 3.2, the number of time slice of the control logic and the resource usage can be reduced significantly if the channel states required for the application are considered. Therefore, the state transitions of control channels are the input of the first step. These states are written as a ***state matrix*** $\tilde{P}$, whose rows represent the states of all control channels at different moments. For example, for the states of the transitions "011"→"100"→"001" →"110", $\tilde{P}$ is written as

$$\tilde{P} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \qquad \tilde{Y} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & X & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{1}$$

In a transition such as "011"→"100", the first control channel needs to be connected to the core input and the pressure value of the core input should be set to '1'. Afterwards, the second and the third control channels need to be connected to the core input with its pressure value set to '0'. In both cases, it is the responsibility of the control logic to connect the corresponding control channels to the core input. These requirements to the control logic can be represented by a ***switching matrix*** $\tilde{Y}$ converted from the state matrix $\tilde{P}$. In this matrix, the rows represent whether the corresponding control channels need to be updated to '0' and '1' alternately. Therefore, these rows are called ***switching patterns***. As an example, the switching matrix of $\tilde{P}$ in (1) is also shown as $\tilde{Y}$. In this matrix, '1' represents that corresponding control output should be connected to the core input. In the transition "100"→"001", when the first channel is updated to '0', the second channel can be updated together with the first channel, or it can be ignored since its state does not change. Accordingly, a don't care 'X' appears. In reality, multiple '1's in a row in $\tilde{Y}$ may not be updated simultaneously, in case this specific multi-channel combination is not implemented. Accordingly, such a row needs to be split into time slices so that the corresponding channels are updated by several operations. To reduce the overall number of time slices, the multi-channel combinations need to be selected carefully.

In a general case, assume that the switching matrix is written as

$$\tilde{Y} = \begin{bmatrix} Y_0 \\ Y_1 \\ \cdots \\ Y_{M-1} \end{bmatrix} = \begin{bmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,N-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,N-1} \\ \cdots & \cdots & \ddots & \cdots \\ y_{M-1,0} & y_{M-1,1} & \cdots & y_{M-1,N-1} \end{bmatrix} \tag{2}$$

where $y_{i,j}$ is a constant taking one of the values '0', '1' or 'X'. M is the number of transitions in which at least a control channel should be switched to '0' or '1'. N is the number of control channels.

As discussed above, a row in $\tilde{Y}$ may contain multiple '1's that cannot be implemented by connecting the corresponding control channels to the core input at the same time. Consequently, the corresponding time slot of switching these control channels needs to be split into several time slices. The objective is that the overall number of time slices implementing the switching matrix $\tilde{Y}$ is reduced. To fulfill this objective, the potential multi-channel switching combinations need to be examined.

For N control channels, there are $2^N - 1$ possible combinations to form multi-channel scheme, defined by the ***multiplexing matrix*** $\tilde{X}$ with N columns, as

$$\tilde{X} = \begin{bmatrix} X_0 \\ X_1 \\ \ldots \\ X_{2^N-2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \ddots & \cdots \\ 1 & 1 & 1 & 1 & \cdots & 1 \end{bmatrix} \tag{3}$$

where each row represents a possible combination of control channels to form the multi-channel switching. If an item $x_{i,j}$ in $\tilde{X}$ is '1', the corresponding control channel is included in the multi-channel switching combination.

Since the objective of multi-channel switching is to select proper combinations of rows in $\tilde{X}$ to implement the switching matrix $\tilde{Y}$, a ***selection matrix*** $\tilde{T}$ of M rows and $(2^N - 1)$ columns is defined as follows

$$\tilde{T} = \begin{bmatrix} t_{0,0} & t_{0,1} & \cdots & t_{0,2^N-2} \\ t_{1,0} & t_{1,1} & \cdots & t_{1,2^N-2} \\ \cdots & \cdots & \ddots & \cdots \\ t_{M-1,0} & t_{M-1,1} & \cdots & k_{M-1,2^N-2} \end{bmatrix} \tag{4}$$

where the ith row defines which rows in $\tilde{X}$ are selected to implement the switching pattern in the ith row of $\tilde{Y}$ in (2).

In a row in (2), if an item $y_{i,k}$ is '1', meaning that this control channel must be activated once, it must be covered by at least one of the rows in $\tilde{X}$ that has a '1' at the corresponding column. This constraint can be expressed as

$$\sum_{j=0}^{j=2^N-2} t_{i,j} x_{j,k} \begin{cases} \geq 1, & y_{i,k} = 1 \\ = 0. & y_{i,k} = 0 \end{cases} \quad \forall i = 0, \ldots M-1,\ k = 0, \ldots N-1 \tag{5}$$

where $x_{i,j}$ and $y_{i,k}$ are given constants. $t_{i,j}$ are 0-1 variables whose values are determined by a solver.

In a control logic, the maximum number of allowed control patterns is usually given or constrained by the number external pressure sources as a constant $Q_{cw} = 2^{\lceil log_2 N \rceil}$ and usually $Q_{cw} \ll 2^N - 1$. Accordingly, for each row in $\tilde{X}$, a 0-1 variable l_i is defined to indicate whether the corresponding combination is selected and the total number of selected combinations should be no larger than Q_{cw}, constrained as

$$\sum_{i=0}^{2^N-2} l_i \leq Q_{cw}. \tag{6}$$

If the j row in $\tilde{X}$ is not selected so that $l_j = 0$, all the selection variables in the j column in $\tilde{T}$ must be set to 0, constrained as

$$t_{i,j} \leq l_j, \quad \forall i = 0, \ldots M-1,\ j = 0, \ldots 2^N - 2. \tag{7}$$

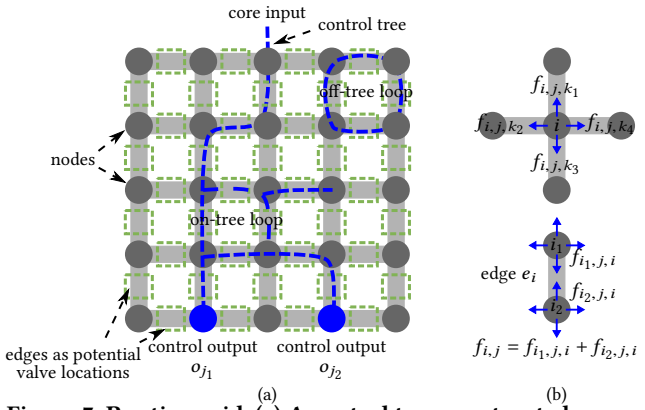


Figure 7: Routing grid. (a) A control tree constructed on a routing grid. (b) Modeling variables representing flow volumes and directions for control tree construction.

Since a row in $\tilde{T}$ represents which multi-channel switching combinations from $\tilde{X}$ are selected to implement the switching patterns in the corresponding row in $\tilde{Y}$, the number of '1's in this row in $\tilde{T}$ represents the required number of time slices. To minimize the total number of slices, the following optimization problem can be solved

$$\text{minimize} \quad \sum_{i=0}^{M-1} \sum_{j=0}^{2^N-2} t_{i,j} \tag{8}$$

$$s.t. \quad (5),\ (6),\ (7). \tag{9}$$

After solving (8)–(9), the combinations of channels to implement multi-channel switching are determined by the values of l_i. The values of $t_{i,j}$ specify how the switching patterns in $\tilde{Y}$ can be realized by these control patterns to reduce the overall switching time.

For an application, the number of rows in the switching matrix $\tilde{Y}$ might be large, making (8)–(9) very difficult to solve. In practice, many rows in the switching matrix $\tilde{Y}$ might be equal. For example, a typical application contains many mixing operations, which use only a few switching patterns repeatedly. In the proposed method, these rows are merged and the number of merged rows are multiplied with $t_{i,j}$ in (8) to reduce the scale of the problem. Another deployed technique to reduce the scale of (8)–(9) is that in the multiplexing matrix, the maximum number of channels that are allowed to switch simultaneously is constrained to a given number. This is acceptable because the case that a large number of channels are updated simultaneously is not common in reality. In the experiments, this maximum number is set to 3.

4.2 Constructing control logic on a general routing grid

After determining the multi-channel switching scheme, the values of the l_i carry the information which control channels should be connected to core input together. Since the states of control channels are the same as those at the outputs of the control logic, it is then the task of the control logic to generate correct patterns at its outputs to drive control channels.

When multiple control outputs are updated by a control pattern, a control path should be constructed for each of them. Since these paths can also share segments with each other, a ***control tree*** is constructed, as illustrated in Fig. 6. In reality, the control tree can be very complex, even with many branches dangling in the middle of the control tree.

In the proposed method, a general routing grid is used as virtual guide to construct control trees. Such a grid is composed of a set of horizontal and vertical edges, and edges join other edges at nodes. On this routing grid, a path can be viewed as a series of consecutive connected edges. On each edge, a control valve can be built. If no valve is built, but the edge still appears in the final control logic, it then always connects the two nodes at its ends. If in the end an edge does not appear in the control logic, the two nodes at the ends of this edge are always not connected directly. An example routing grid is shown in Fig. 7(a), where o_{j_1} and o_{j_2} are control outputs.

For an edge e_i on the routing grid, a 0-1 variable e_i^{exist} is used to indicate whether the edge appears in the control logic. In addition, a 0-1 variable v_i^{exist} is used to indicate whether a control valve v_i is built on the edge e_i. If a control valve appears on the control tree to drive the control output o_j, it must be switched open when o_j is activated. This open/closed state is denoted by the 0-1 variable $v_{i,j}^{state}$. The relation between these variables can be written as

$$v_{i,j}^{state} \le v_i^{exist} \le e_i^{exist}, \quad \forall e_i \in \mathcal{E}, c_j \in C \tag{10}$$

where $\mathcal{E}$ is the set of all edges on the routing grid, and C is the set of all control patterns.

To construct a control tree in the control logic, the connection state of an edge should be defined first. In two scenarios, an edge connects the two nodes at its ends: 1) an edge appears in the control logic but there is no valve built on it; 2) a valve is built on an edge that appears in the control logic, and the valve is open due to the control port driving it. Such an edge is called a ***connection edge*** in the following. If the 0-1 variable $c_{i,j}$ is used to indicate whether the edge e_i connects the nodes at its two ends when the jth control pattern applied, $c_{i,j}$ can be constrained as

$$c_{i,j} = e_i^{exist} - v_i^{exist} + v_{i,j}^{state}. \tag{11}$$

With the connection states of edges defined above, the control tree can thus be described accordingly. To construct a control tree, the idea is to use the concept of a flow from the core input. The flow fills the edges it passes and only reaches the control outputs that should be activated.

For each node n_i in the routing a grid, a flow value $f_{i,j,k}$ is defined with respect to control pattern c_j and the kth edge that is incident to n_i directly, as shown in Fig. 7(b). If a flow enters a node, the flow value is negative. If it leaves a node, its flow is positive. Since a node only connects edges but cannot store a flow volume, for node n_i that does not correspond to the core input or a control output that should be activated, the relation between the flow variables can be written as

$$\sum_{e_k \in \mathcal{E}_i} f_{i,j,k} = 0 \tag{12}$$

where $\mathcal{E}_i$ is the set of edges incident to node n_i directly.

For each edge e_i, the variable $f_{i,j}$ is defined to represent whether the edge stores one unit of the flow, and is determined by the flows entering the edge from the two nodes at its ends and could be one only when the edge is open. The relation is expressed as

$$f_{i,j} = \sum_{n_k \in \mathcal{N}_i} f_{k,j,i} \le c_{i,j} \tag{13}$$

where $\mathcal{N}_i$ is the set of nodes at the ends of edge e_i, and $f_{k,j,i}$ is a flow value to e_i.

To form a control tree from the core input to control output o_j, the nodes in-between must function as connecting points. A 0-1 variable $n_{i,j}$ is defined to represent whether node n_i is in the tree or not. For a node in the tree, at least an edge incident to it should be filled by the flow. The connection condition for node n_i is

$$n_{i,j} \le \sum_{e_k \in \mathcal{E}_i} f_{k,j} \le 4 \cdot n_{i,j} \tag{14}$$

where $\mathcal{E}_i$ is the set of edges connecting to n_i.

Since the core input needs to provide sufficient flow to fill the edges in the control tree and the flow must only reach the current control output o_j, the following constraints can be established.

$$\sum_{e_k \in \mathcal{E}_i} f_{i,j,k} > 0, \quad n_i = n_{core} \tag{15}$$

$$\sum_{e_k \in \mathcal{E}_i} f_{i,j,k} < 0, \quad n_i \text{ represents an opened output } o_j. \tag{16}$$

For the outputs that are closed, the nodes representing them should not appear on the control tree. Since an edge that is in the connection state makes its two nodes share the same status, $n_{i,j}$ needs to be constrained as

$$n_{i,j} = 0, \ \forall n_i \text{ representing a closed output} \tag{17}$$

$$c_{i,j} - 1 \le n_{k_1,j} - n_{k_2,j} \le 1 - c_{i,j}, \ \forall e_i \in \mathcal{E} \tag{18}$$

where $n_{k_1,j}$ and $n_{k_2,j}$ are the two nodes of e_i, and $\mathcal{E}$ is the set of all edges.

The constraints above can be used to generate a control tree shown in Fig. 7(a). These constraints do not prohibit an on-tree loop such as that in Fig. 7(a) from happening. The existence of on-tree loops, however, does not affect the correct function of the control logic. The off-tree loop in Fig. 7(a) is excluded by the constraints (10)–(18). However, these constraints are only sufficient conditions to construct a control tree. Off-tree loops can still appear, provided that they do not activate the current control output.

The flow value $f_{i,j}$ defines whether an edge is required to control an output. If a valve appears on the edge, its connection to the control ports needs to be determined, so that it can be switched correctly by an external pressure source. Assume there are N_p control ports. Since a control valve can be connected to any of these N_p ports, for the control valve v_i, N_p 0-1 variables $p_{i,1}, p_{i,2}, \ldots p_{i,N_p}$ are defined. The variable $p_{i,k}$ represents whether control valve v_i is connected to the kth control port. Since a control valve can only be controlled by one port when a control valve exists on an edge, these variables are constrained as

$$\sum_{k=1}^{N_p} p_{i,k} = v_i^{exist}. \tag{19}$$

For the jth control tree, corresponding to the jth control pattern, assume the states of the control ports are denoted by 0-1 variables $s_{j,1}, s_{j,2}, \ldots s_{j,N_p}$. For the control valve v_i, its state corresponding to the jth control pattern is written as $v_{i,j}^{state}$. Since all the valves controlled by the same control port must have the same state in a control pattern, the valve states can be constrained as

$$p_{i,k} - 1 \le v_{i,j}^{state} - s_{j,k} \le 1 - p_{i,k}, \quad \forall e_i \in \mathcal{E}, \ k \in \{1, \ldots N_p\} \tag{20}$$

where $\mathcal{E}$ is the set of all edges.

In a control logic, the control patterns should be different in activating different control outputs or their combinations. Therefore, when regarded as binary numbers, the values of the control patterns are different from each other. This condition can be specified as

$$B_j = 2^0 \cdot s_{j,1} + 2^1 \cdot s_{j,2} + \cdots + 2^{N_p - 1} \cdot s_{j,N_p} \tag{21}$$

$$B_{j_1} - B_{j_2} \le -1 + My, \ \forall j_1 \ne j_2 \tag{22}$$

$$B_{j_1} - B_{j_2} \ge 1 - (1 - y)M, \ \forall j_1 \ne j_2 \tag{23}$$

where M is a large number, y is an intermediate 0-1 variable, where $y = 1$ if and only if $B_{j1} > B_{j2}$ and $y = 0$ if and only if $B_{j1} < B_{j2}$.

The constraints described in this section are very general. To implement multi-channel switching, a control tree needs to activate multiple control outputs simultaneously. Accordingly, these active outputs can simply be enabled by adding constraints similar to (16), so that the control tree drives multiple control outputs at the same time.

To implement backup paths for fault tolerance, a path needs to be identified from a control tree first. For example, in Fig. 7, the on-tree loop does not need to be duplicated, because the direct path between the core input and the control output is already sufficient for state updating. To identify a path in the control tree, a model similar to that used to identify the control tree can be deployed. In constructing the control tree, the edges are chained one after another. To constrain a path instead of a tree, the only change to be made is that a node in the routing grid that represents neither the core input nor the control output is only allowed to connect exactly two edges, in contrast to (14), where more than two edges can be connected to a node to allow more freedom to the patterns of control ports. For fault tolerance, two identified control paths that are backup to each other should not share any parts on the routing grid. This constraint can be specified as that the edges covered by the paths should not overlap, so that the variables indicating that an edge belonging to these paths should be 1 only for one of the fault-tolerant paths. These constraints are similar to those for constructing control trees discussed above, and are not discussed in detail due to limited space.

With the constraints defined, the control logic can be constructed by creating a control tree for each control pattern and solving the resulting ILP problem as

$$\text{minimize} \quad \sum_{i=0}^{|\mathcal{E}|-1} v_i^{exist} \tag{24}$$

$$s.t. \quad (10)\text{–}(23). \tag{25}$$

To improve the efficiency of the formulation (24-25), two heuristic techniques have been applied. First, the control ports are only allowed to control the valves in the rows and columns that are neighboring to them in the routing grid. Second, the routing grid is partitioned into sub-blocks and the formulation (24-25) is applied to each sub-block to solve the problem hierarchically. In addition, pressure degradation in control trees has also be considered, which is not explained in detail due to space limit.

5 SIMULATION RESULTS

The proposed method was implemented in C/C++ and tested by using 2.4 GHz CPU with 32GB memory. We demonstrate the results of three applications that are CPA (Colorimetric Protein Assay) used in RA30 chip from [4], IVD (Int-Vitro Diagnostics) applied in CPA chip from [4] and mRNA chip from [19]. In addition, three randomly generated sequences of switching patterns R0, R1, and R2 are tested to demonstrate the characteristics of the proposed method further.

The information of these test cases are shown in Table 1, where the column #M shows the numbers of mixers used by the applications, and #V is the number of flow valves and thus the number of control channels. The numbers of control ports to implement the control variables are reported in the column #P. The numbers of states of flow valves in executing the corresponding applications are reported in the column #F and the numbers of switching patterns corresponding to the row of the switching matrix $\tilde{Y}$ in (2) are reported in the column #Y. After merging equivalent rows of switching patterns as described at the end of Section 4.1, the numbers of independent patterns used in (8)–(9) are shown in the column #Y′.

As discussed previously, without multi-channel switching, the '1's in a switching matrix must be updated individually. For the applications in the experiments, the total numbers of time slices are reported in the column T_s in Table 1. With multiple-channel switching, these numbers are reduced significantly in most cases, as shown in the column T_m.

Table 1: Results of Multiplexing and Fault Tolerance

App.	#M	#V	#P	#F	#Y	#Y′	T_s	T_m	R	#Q	V_c	t_r
RA30	2	19	10	10221	13408	86	27025	15247	44%	17	137	1309
CPA	3	25	10	2941	1409	92	4198	1742	59%	22	179	2159
mRNA	3	37	12	5361	1403	52	4464	1597	65%	20	212	3173
R0	1	22	10	5000	6684	153	6891	6799	1%	26	274	3999
R1	2	27	10	6000	8013	244	14334	9776	32%	28	330	4040
R2	3	48	12	7000	9372	325	26058	11781	54%	51	812	10586

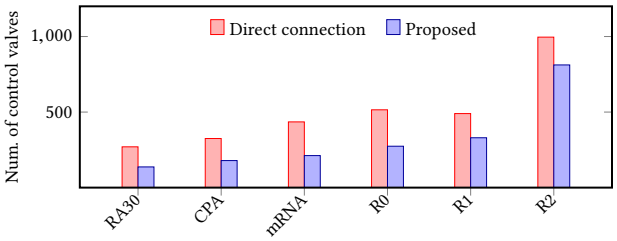


Figure 8: Comparison of numbers of control valves between direct connection and with multi-channel switching.

The ratio of reduction of these switching activities can reach up to 56%, as shown in the column R, while 1% in case R1 is because the channel patterns in this case is totally random and uncorrelated which is uncommon in real applications. The numbers of control patterns used in the control logic are shown in the column #Q, which are larger than the numbers of control channels in the column #V due to the additional control patterns for multi-channel switching for cases R1, R2 and R3, while being slightly smaller in cases RA30, CPA and mRNA since several flow valves in these cases always activate simultaneously with other valves so that their control patterns are shared. It can be observed that with a limited increase of the number of control patterns, a significant reduction of switching time slices from T_s to T_m can be achieved. Finally, the numbers of control valves in the control logic are reported in the column V_c and the CPU time to generate the control logic by the proposed method is reported in the column $t_r(s)$.

To verify the efficiency of the automatically generated control logic, the numbers of control valves are compared with the cases with direct connection from core input to the control channels as shown in Fig. 3. The results of this comparison is illustrated in Fig. 8. Obviously the control logic from the proposed method requires fewer control valves. Meanwhile the number of time slice is smaller as demonstrated in the columns T_s and T_m in Table 1.

In determining multi-channel switching patterns, the maximum number of control channels that can be switched together is bound to a given number to increase the implementation efficiency. The reduction ratios of switching time with different maximum number of channels being switched together are shown in Fig. 9. As expected, the reduction ratios increase as the numbers of multiple channels switching together increase. However, a further increase from 3 to 4 does not lead to significant performance improvement, justifying the bound set in the proposed method.

6 CONCLUSION

In this paper, a general framework to generate control logic for flow-based microfluidic biochips has been proposed. By introducing the multi-channel switching scheme, the time required for switching valves can be reduced significantly. Compared with the traditional design with direct connections, the proposed control logic also uses fewer control valves. Furthermore, independent backup control paths have also been introduced to improve fault tolerance of the automatically generated control logic. The proposed method is based on ILP formulation. In the future work heuristic techniques will be introduced to improve its scalability.

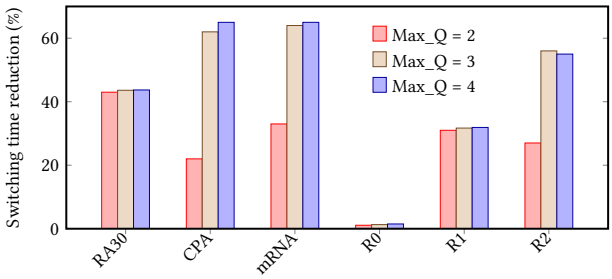


Figure 9: Reduction of total channel switching time under different multiplexing distances

ACKNOWLEDGMENT

Y. Zhu, B. Li and U. Schlichtmann were supported by Deutsche Forschungsgemeinschaft (DFG) through TUM International Graduate School of Science and Engineering (IGSSE). T.-Y. Ho was supported in part by the Technical University of Munich – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

REFERENCES

[1] J. M. Perkel, "Microfluidics: Bringing new things to life science," *Science*, vol. 322, no. 5903, pp. 975–977, 2008.
[2] I. E. Araci and S. R. Quake, "Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves," *Lab Chip*, vol. 12, pp. 2803–2806, 2012.
[3] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga, "Architectural synthesis of flow-based microfluidic large-scale integration biochips," in *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embed. Sys.*, 2012, pp. 181–190.
[4] C. Liu, B. Li, H. Yao, P. Pop, T.-Y. Ho, and U. Schlichtmann, "Transport or store? Synthesizing flow-based microfluidic biochips using distributed channel storage," in *Proc. Design Autom. Conf.*, 2017, pp. 49:1–49:6.
[5] Q. Wang, H. Zou, H. Yao, T.-Y. Ho, R. Wille, and Y. Cai, "Physical co-design of flow and control layers for flow-based microfluidic biochips," *IEEE J. Technol. Comput. Aided Design*, vol. 37, no. 6, pp. 1157–1170, 2018.
[6] C.-X. Lin, C.-H. Liu, I.-C. Chen, D. T. Lee, and T.-Y. Ho, "An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips," in *Proc. Design Autom. Conf.*, 2014, pp. 141:1–141:6.
[7] A. Grimmer, Q. Wang, H. Yao, T.-Y. Ho, and R. Wille, "Close-to-optimal placement and routing for continuous-flow microfluidic biochips," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2017, pp. 530–535.
[8] K. Hu, F. Yu, T.-Y. Ho, and K. Chakrabarty, "Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 10, pp. 1463–1475, 2014.
[9] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty, "Control-layer routing and control-pin minimization for flow-based microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 1, pp. 55–68, 2017.
[10] H. Yao, T.-Y. Ho, and Y. Cai, "PACOR: practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips," in *Proc. Design Autom. Conf.*, 2015, pp. 142:1–142:6.
[11] Q. Wang, S. Zuo, H. Yao, T.-Y. Ho, B. Li, U. Schlichtmann, and Y. Cai, "Hamming-distance-based valve-switching optimization for control-layer multiplexing in flow-based microfluidic biochips," in *Proc. Asia and South Pacific Des. Autom. Conf.*, 2017, pp. 524–529.
[12] K. S. Elvira, X. C. i Solvas, R. C. R. Wootton, and A. J. deMello, "The past, present and potential for microfluidic reactor technology in chemical synthesis," *Nature Chemistry*, no. 5, pp. 905–915, 2013.
[13] L. M. Fidalgo and S. J. Maerkl, "A software-programmable microfluidic device for automated biology," *Lab Chip*, vol. 11, pp. 1612–1619, 2011.
[14] T.-M. Tseng, B. Li, M. Li, T.-Y. Ho, and U. Schlichtmann, "Reliability-aware synthesis with dynamic device mapping and fluid routing for flow-based microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 12, pp. 1981–1994, 2016.
[15] C. Liu, B. Li, B. B. Bhattacharya, K. Chakrabarty, T.-Y. Ho, and U. Schlichtmann, "Testing microfluidic fully programmable valve arrays (FPVAs)," in *Proc. Design, Autom., and Test Europe Conf.*, 2017, pp. 91–96.
[16] J. Melin and S. Quake, "Microfluidic large-scale integration: the evolution of design rules for biological automation," *Annu. Rev. Biophys. Biomol. Struct.*, vol. 36, pp. 213–231, 2007.
[17] I. E. Araci, P. Pop, and K. Chakrabarty, "Microfluidic very large-scale integration for biochips: Technology, testing and fault-tolerant design," in *Proc. Int. Euro. Test Symp.*, 2015, pp. 1–8.
[18] W.-L. Huang, A. Gupta, S. Roy, T.-Y. Ho, and P. Pop, "Fast architecture-level synthesis of fault-tolerant flow-based microfluidic biochips," in *Proc. Design, Autom., and Test Europe Conf.*, 2017, pp. 1671–1676.
[19] J. S. Marcus, W. F. Anderson, and S. R. Quake, "Microfluidic single-cell mRNA isolation and analysis," *Analytical Chemistry*, vol. 78, no. 9, pp. 3084–3089, 2006.