MultiControl: Advanced Control Logic Synthesis for Flow-Based Microfluidic Biochips

Ying Zhu, Xing Huang, Bing Li, Tsung-Yi Ho, *Senior Member, IEEE,* Qin Wang, Hailong Yao, *Senior Member, IEEE,* Robert Wille, *Senior Member, IEEE,* and Ulf Schlichtmann, *Senior Member, IEEE*

***Abstract*—Flow-based microfluidic biochips are one of the most promising platforms used in biochemical and pharmaceutical laboratories due to their high efficiency and low costs. Inside such a chip, fluids of nanoliter volumes 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 by a control logic using 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 present the first automatic synthesis approach for a control logic that is able to switch multiple control channels simultaneously. Moreover, we propose the first fault-aware design in control logic by introducing backup control paths to maintain the correct function even when manufacturing defects occur. The construction of control logic is achieved by a highly efficient framework based on Particle Swarm Optimization, Boolean logic simplification, grid routing, together with mixing multiplexing. Simulation results demonstrate that the proposed multi-channel switching mechanism leads to fewer valve-switching times and lower total logic cost, while realizing fault tolerance for all control channels.**

***Index Terms*—Flow-based microfluidic biochips, control logic, fault tolerance, channel multiplexing.**

I. 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 the need for human intervention. To improve the execution efficiency of experiments, biochemical industry has achieved a remarkable advance by providing system-in-a-package solutions, where experiments can be performed within a compact system completely. Although this system integration improves experiment efficiency significantly compared with traditional biochemical laboratories, 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 [2].

To overcome the shortcomings of the systems above, flow-based microfluidic biochips have been investigated intensely in the past decade [2], [3]. In such a chip, a large number of devices, e.g., mixers and detectors, are constructed. These devices are connected by microchannels (also referred to as flow channels) to transport intermediate experiment results. The transportation of these results is controlled by microvalves, which are tiny switches built on top of flow channels [4].

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 microchannels on a substrate [5]. 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 [6]–[12] and the routing of flow channels in [13]–[15]. Furthermore, test methods for defect detection after manufacturing have also been proposed in [16], [17].

Compared with integrated circuits, biochips, however, exhibit some specific features. Besides flow channels that are used to transport fluid samples, valves need to be driven by external air/fluid-pressure patterns to change their states. The schematic of a biochip is shown in Fig. 1(a) and 1(b), where the control channel is built on top of the flow channel. An air/fluid pressure through the control channel squeezes the flow channel to block the movement of flow samples. When the air/fluid 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/fluid pressure in the control channel. With valves as the controlling units, complex biochips can be constructed, as shown in Fig. 1(c) [18].

When executing an application, the patterns of air/fluid 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,

A preliminary version of this work was published in the Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2018 [1]. Extensions beyond [1] include a new multiplexing mechanism for peristalsis valves in mixers, an efficient and effective control-architecture synthesis flow based on a hybrid Particle Swarm Optimization, two control-logic reduction strategies, a grid-routing-based control-path construction algorithm, as well as comprehensive experimental evaluations. (*Corresponding author: Xing Huang*)

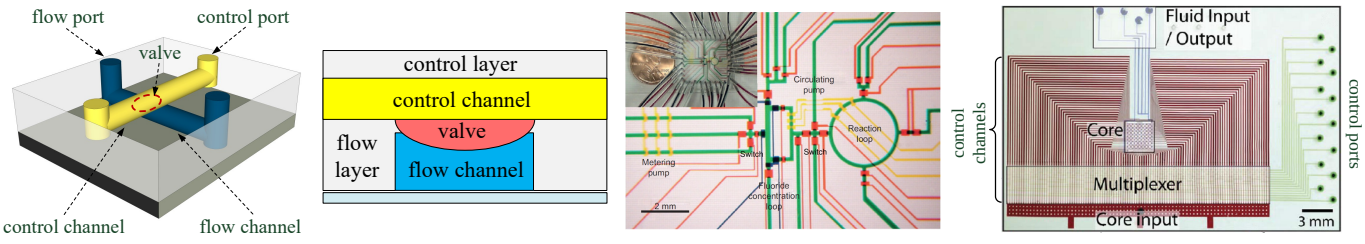
Y. Zhu, B. Li, and U. Schlichtmann are with the Chair of Electronic Design Automation, Technical University of Munich, Germany (e-mail: ying.zhu@tum.de, b.li@tum.de, ulf.schlichtmann@tum.de)

X. Huang and T.-Y. Ho are with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan (e-mail: xing.huang1010@gmail.com, tyho@cs.nthu.edu.tw)

X. Huang is also with the College of Mathematics and Computer Science, Fuzhou University, Fuzhou, China (e-mail: xing.huang1010@gmail.com)

Q. Wang and H. L. Yao are with the Department of Computer Science and Technology, Tsinghua University, Beijing, China (e-mail: woodythu@163.com, hailongyao@tsinghua.edu.cn)

R. Wille is with the Institute for Integrated Circuits, Johannes Kepler University Linz, Austria. (e-mail: robert.wille@jku.at)



(a) (b) (c) (d)

Fig. 1: (a) Schematic of flow-based microfluidic biochips. (b) Cross section of (a). (c) A microfluidic biochip with flow channels (green) and control channels (yellow and red) [18] (d) Structure of a complete biochip [19].

related research considering control channel optimization has started to appear. For example, the method in [20] minimizes pressure-propagation delay in control channels to reduce the response time of valves. The lengths of control channels are matched in [21] 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 has not sufficiently been investigated yet. Up to now, only one method has been proposed to consider the reliability of control logic [22], where the order of patterns that are required to control valves is adjusted to reduce the maximum number of switching times 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. The major contributions of this paper are listed as follows:

- We present the first practical problem formulation for control logic design considering both control multiplexing and fault tolerance in flow-based microfluidic biochips. It is of great significance in improving the execution efficiency and reliability of biochips, while minimizing the total cost of a control logic.
- The basic design rules of control logic are examined and a new channel-switching mechanism is proposed for the first time, which can switch *multiple* control channels simultaneously by expressing channel switching patterns with Boolean logic. Together with the newly introduced switching state compression by mixing control, the efficiency of generating the required pressure patterns can be improved significantly.
- 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 new structures of control logic. To the best of our knowledge, this is the first work to examine the design of control logic itself.
- A systematic approach incorporating both multi-channel switching and fault tolerance is presented for automatic control-logic construction, which determines locations of control valves as well as their connections using a hybrid Particle Swarm Optimization (PSO) and an efficient grid-routing algorithm.
- Simulation results confirm that the optimized control logic leads to fewer valve-switching times and lower total logic cost, while fault tolerance for all control channels is implemented successfully.

The rest of this paper is organized as follows. In Section II, the existing structure of control logic design is explained and its limitations are discussed. In Section III, new mechanisms for multi-channel switching and fault-tolerance are presented, and the problem model of control logic design in flow-based microfluidic biochips are formulated. The proposed control logic design methods are then described in detail in Section IV. Simulation results are reported in Section V. Conclusions are drawn in Section VI.

II. 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/fluid pressure generated by the control logic. In Fig. 1(d) an example of a complete biochip from [19] is shown. The flow core of the biochip is located at the center for executing biochemical operations. The control channels surrounding the flow core, the multiplexer, the core input, as well as the pressure sources on the right-hand side together form a control logic to generate pressure patterns to switch valves in the flow core. Due to the cost and the size of the mechanical components, it is not practical to assign each valve an independent pressure source. For example, in the design in Fig. 1(d), 116 valves in the flow core have been implemented. For executing applications, instead of using 116 pressure sources directly, which would be very cumbersome and expensive, only 15 pressure sources are used to generate pressure patterns. In other words, a control logic using time multiplexing is deployed to switch the states of valves.

In Fig. 1(d), the core input at the bottom provides a pressure source that can be switched on or off. On the right, the control ports are connected to external pressure sources to create ***control patterns*** that specify which control channel can be connected to the core input. The multiplexer 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 value is updated to the same as that of the core input. Correspondingly,

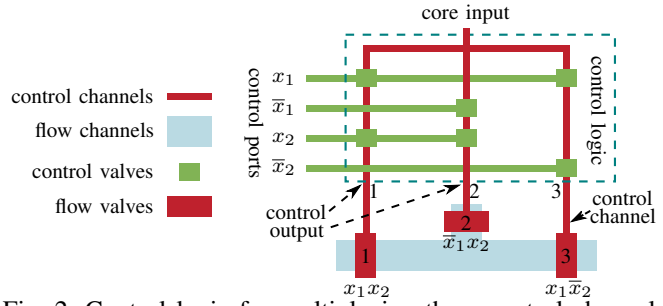


Fig. 2: Control logic for multiplexing three control channels. Control ports x_1, $\overline{x}_1$, x_2, and $\overline{x}_2$ are connected to external pressure sources.

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.

Fig. 2 explains the multiplexing function of the control logic to reduce the number of pressure sources. In this example, four control ports x_1, $\overline{x}_1$, x_2, $\overline{x}_2$ are connected to pressure sources to control the connection of the control channels that drive the three flow valves. In control logic design, the pressure values of the control ports are often complementary [19], [23]. 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. 2. 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 cahnnel is to be connected to the core input to change the valves of the control outputs. 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'. In the following, the terms "control channel" and "control path" will be used interchangeably.

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. 2 to control the three channels. At any moment, only one of them can be true, so that only one control output can be connected to the core input for updating its pressure value. If the target pressure 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. 2. In the control logic in Fig. 2, the number of pressure sources is five, which is larger than the number of control channels. Therefore, the control multiplexing actually requires more pressure 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 with N.

The function of the control logic shown in Fig. 2 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 channel is high and '0' represents the pressure is low. At time $t+1$, assume that the states of the control channels need to be updated to "100". Since the control logic in Fig. 2 only allows one control channel to be connected to the core input 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", "01" and "10", 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 sequentially be set to '1', '0' and '0' to update the pressures in the control channels. For convenience, the time to update all control channels from their states at time t to their 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.

III. Proposed Multiplexing Mechanisms for Control Logic Design and Problem Formulation

The control logic design described above is very effective in reducing the number of pressure sources. However, flow valves are switched sequentially 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 need to be updated is large, the execution time of the application can be prolonged. This disadvantage is due to the fact that only one output can be updated in a time slice. To solve this problem, a new design scheme that allows multiple control outputs to be activated simultaneously will be introduced in the following to improve the efficiency of the 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 multi-channel switching to improve design efficiency.

A. Multi-channel switching

In Fig. 2, only three flow valves are driven by the control logic, though 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. 2,

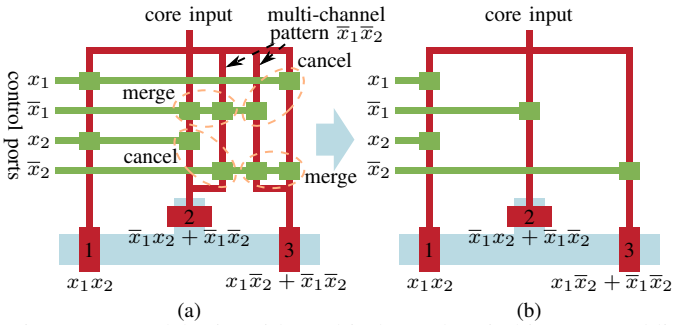


Fig. 3: Control logic with multi-channel switching. (a) Additional control pattern $\overline{x}_1\overline{x}_2$ is used to update control channels 2 and 3 simultaneously. (b) Simplified control logic after valve merging and canceling.

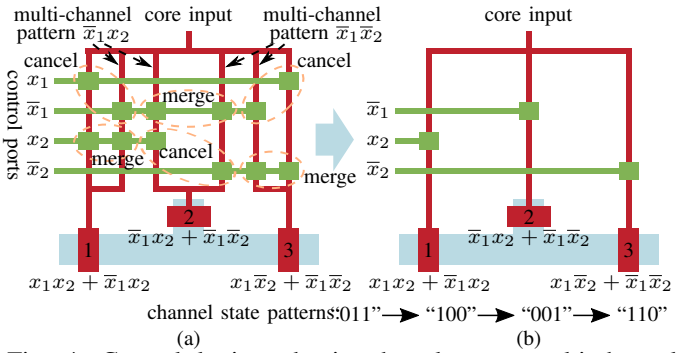


Fig. 4: Control logic reduction by alternate multi-channel switching. (a) Control pattern $\overline{x}_1x_2$ is used to update control channels 1 and 2 simultaneously and control channel 2 has no individual control pattern. (b) Simplified control logic.

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. 3(a). We call this "multi-channel switching". In this augmented design, both channels 2 and 3 are connected to the core input through the newly added control paths. Consequently, in the transition from "011"→"100", the number of time slices can be reduced by 1.

In Fig. 3(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. Consequently, the control logic can be simplified as shown in Fig. 3(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. 3(b) the number of valves has been reduced from 10 to 4 compared with Fig. 3(a). Compared with the original control logic in Fig. 2, 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. 3(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 total number of 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. 2, where only single-channel switching is possible.

B. Logic reduction by alternate multi-channel switching for given applications

In the case in Fig. 3(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 the 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. 3(a) can be spared to implement multi-channel switching between channels 1 and 2, as shown in Fig. 4(a).

In Fig. 4(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 suitable only for a given application. But the control logic itself can be simplified and the switching times of valves in executing the application can be reduced.

After the merging and canceling operations are applied to the case in Fig. 4(a), only three control valves are left in the design, as shown in Fig. 4(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

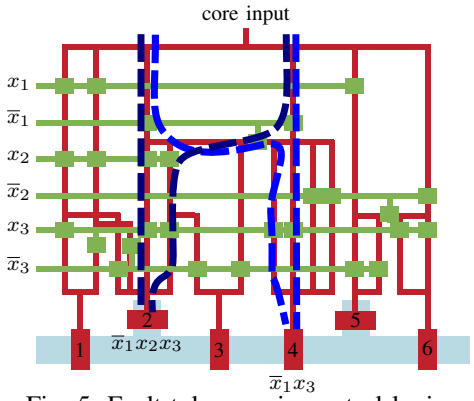


Fig. 5: Fault tolerance in control logic.

"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 of the chip is independent of whether the state of the second channel is updated or not, similar to a "don't care" channel in logic design. In the design in Fig. 4(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.

C. Fault tolerance in control logic

In Fig. 4(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 [24]. 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.

Fig. 5 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.

D. Problem formulation

Based on the new mechanisms discussed above, the control-logic design considering control multiplexing and fault-tolerance can be formulated as follows:

Input: The states of all flow valves/control channels at every moment in a given biochemical application.

Output: An optimized control logic supporting multi-channel switching and fault tolerance.

Objective: (1) Minimize the number of time slices for channel switching, (2) minimize the number of control valves, and (3) minimize the total control-channel length.

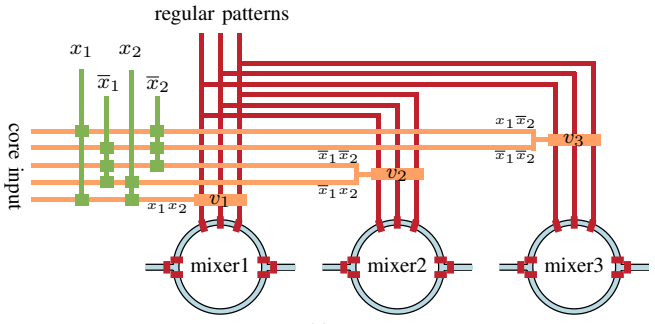
IV. A General Framework for Control Multiplexing and Fault Tolerance

To generate a control logic supporting multi-channel switching and fault tolerance, a general framework including two major steps is adopted in this section. First, the given control channel states are converted to channel switching patterns. Then control channels that can be enabled simultaneously are identified to reduce the total number of time slices. In the second step, control channels are constructed to meet the multi-channel switching and fault-tolerance requirements and thus generate the final control logic.

A. Switching states compression by mixing multiplexing

The complexity of control logic is affected by the flow-valve states to be generated. Generally more valve states lead to more control channels and valves. In practice, a large number of valve states are actually generated by mixers. As shown in Fig. 6(a), each of the three mixers has three flow valves at the top to create a circular flow for peristalsis mixing. This function requires these valves to be switched with a high frequency within a given time period. The flow-valve states need to repeat a given pattern series, e.g., "010"→"011"→"001"→"101"→"100"→"110" [23]. Assume that each of the three mixers in Fig. 6(a) is activated by this pattern series once, but at a different time. An exemplary flow-valve switching states are thus shown as in Fig. 6(b), where the bold patterns highlight the valves need to be switched (in total 21 states should be switched in this example).

In a mixer, the flow valves for peristalsis are only used to create a circular flow with the given pattern series, no matter from which pattern the series starts. In other words, the switching series can be rotated, as long as the whole pattern series is repeated. To compress switching times, we take advantage of this feature by driving all mixers with the same peristalsis patterns, as shown in Fig. 6(a). The three control ports at the top of this structure provide a repeating regular pattern series for all the peristalsis valves. The real connection of these ports to the peristalsis valves in the mixers are controlled by newly introduced valves v_1, v_2 and v_3. Therefore, the switching states in Fig. 6(b) can be converted into Fig. 6(c), where v_1, v_2 and v_3 are opened with a low pressure in their control channels when the mixers start, and they are closed with a high pressure when the mixers stop. Since the regular patterns are shared by all mixers and switch very often, they are generated by external pressure sources directly. The control logic only needs to generate the corresponding states for v_1, v_2 and v_3. Compared with the



(a)

mixer1	mixer2	mixer3
000	000	000
000	000	000
010	000	000
011	000	000
001	000	000
101	**010**	000
100	**011**	000
110	**001**	000
000	**101**	**010**
000	**100**	**011**
000	**110**	**001**
000	**000**	**101**
000	000	**100**
000	000	**110**
000	000	**000**
000	000	000

(b)

regular patterns	$v_1v_2v_3$
000	111
000	111
010	**011**
011	011
001	**011**
101	**001**
100	001
110	**001**
010	**100**
011	100
001	**100**
101	**110**
100	110
110	**110**
000	**111**
000	111

(c)

Fig. 6: Switching states compression by mixing multiplexing. (a) Structure of mixing multiplexing. (b) Switching states when peristalsis valves in mixers are controlled separately. (c) Switching states with mixing multiplexing.

original direct control of peristalsis valves in Fig. 6(b), the control logic only needs to produce 5 switching activities, fewer than a fourth of the switching activities in Fig. 6(b).

To implement mixing multiplexing, the original switching states are examined. For each mixer, a new valve is created on the control paths to its peristalsis valves. These valves are considered control valves and the original peristalsis valves are removed from the control patterns. Consequently, the control logic can generate the control patterns such as x_1x_2, $\overline{x}_1x_2$ and $x_1\overline{x}_2$ to control v_1, v_2 and v_3, respectively. In addition, mixers can be activated by multi-channel switching. For example, mixer2 and mixer3 in Fig. 6(c) can be activated by the control pattern $\overline{x}_1\overline{x}_2$ simultaneously. This concept of mixing multiplexing is a pre-processing step for control logic construction, and it scales well as the number of mixers in the chip increases, since for each mixer only the control patterns of one valve need to be generated, which only mark the starting and stopping time of the mixer and thus do not switch often.

B. Computation of multi-channel switching scheme

As discussed in Section III-B, the number of time slices of the control logic and the resource usage can be reduced significantly if the control channel states required for the application are considered. 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*** $\tilde{Y}$ derived from the state matrix $\tilde{P}$. In this matrix, a '1' represents that the corresponding control channel is connected to the core input and its state is updated to the same as that of the core input; a '0' indicates no update of the corresponding control channel. Therefore, these rows are called ***switching patterns***. As an example, the switching matrix of $\tilde{P}$ in (1) is also shown as $\tilde{Y}$. Note that 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. Therefore, 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 one channel should be switched. N is the number of control channels.

As discussed above, a row in $\tilde{Y}$ may contain multiple '1's that cannot be implemented simultaneously. 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 ***multi-***

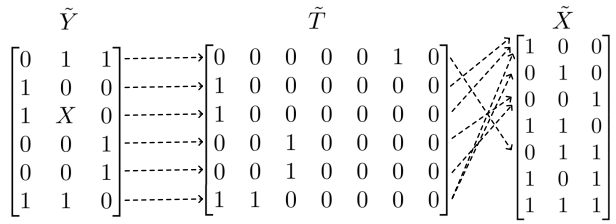


Fig. 7: Multiplexing matrix and a feasible selection matrix corresponding to the switching matrix in (1).

plexing matrix $\tilde{X}$ with N columns, as

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

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

For example, Fig. 7 shows the multiplexing matrix and a feasible selection matrix corresponding to the switching matrix in (1). There are a total of seven channel combinations and four of them are selected to implement the switching patterns defined in $\tilde{Y}$. Note that the last switching pattern in $\tilde{Y}$, i.e., '110', is split into two time slices to update the states of the first two control channels. More specifically, the channel combinations '100' and '010' are selected to update the states of the two channels sequentially.

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

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

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

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

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 times.

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.

C. Control-logic construction

With the ILP model formulated above, control channels that need to be switched independently/simultaneously are determined, and it is then the task to construct a control logic that can meet the channel-switching requirements while taking the fault-tolerance design into account. Accordingly, in this subsection, we present an efficient heuristic for control-logic construction, which consists of two major steps: 1) control-architecture synthesis and 2) control-path construction.

1) Control-architecture synthesis

After computing the multi-channel switching scheme in Section IV-B, the 0-1 variables l_i, $i = 0, 1, \ldots, 2^N - 2$ carry the information of all the selected multiplexing combinations.

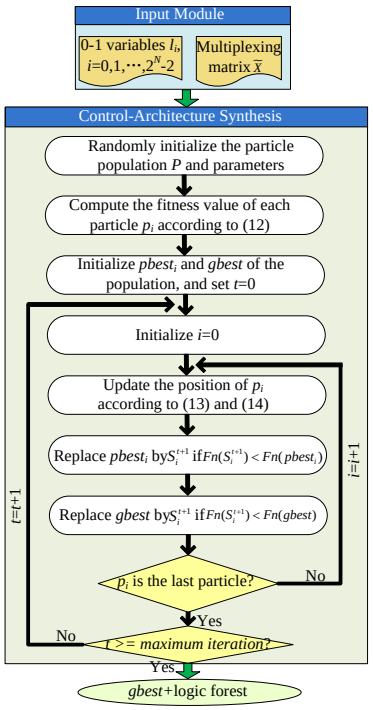


Fig. 8: Flow chart of the proposed control-architecture synthesis flow.

Moreover, the number of available control ports is initialized to $2 * \lceil \log_2 N \rceil$, which can generate a total of $2^{\lceil log_2 N \rceil}$ different control patterns accordingly. The major goal of control-architecture synthesis is then to assign each control channel exact control patterns, so that the multi-channel switching scheme determined in Section IV-B can be effectively implemented and the total cost of the control logic is minimized. To this end, we apply the heuristic Particle Swarm Optimization [25], which was developed through simulation of the social behaviors of bird flocks, and has been found to be robust in solving complex engineering and optimization problems [26], [27]. Fig. 8 shows the flow chart of the proposed control-architecture synthesis flow.

Consider a particle population P, each particle p_i is associated with a velocity vector V_i representing the flying direction and a position vector S_i indicating the current solution in the search space. A fitness function is applied for particles to evaluate the quality of their positions. Moreover, a local best position $pbest_i$ of p_i in its search history is tracked, and a global best position $gbest$ with the best assessment of all particles is also recorded. The whole population is then updated by iteratively changing the position of each particle, based on its flying velocity as well as the corresponding experience perception (i.e., exchange information with $pbest_i$ and $gbest$). After a certain number of iterations, the global best position of the population is selected as the final solution.

Particle Encoding and Evaluation Mechanism: In the proposed method, the position vector S_i of a particle $p_i \in P$ is encoded as

$$cp(i_1), cp(i_2), \ldots, cp(i_j),\ 0 \leqslant i_j \leqslant 2^N - 2 \ and \ l_{i_j} = 1 \quad (10)$$

$$s.t. \quad \forall s, t \ with \ s \neq t,\ cp(i_s) \neq cp(i_t) \quad (11)$$

where the value of each $cp(i_j)$ is set to a constant k ($0 \leqslant k < 2^{\lceil log_2 N \rceil}$), indicating that the k-th control pattern is assigned to the multiplexing combination l_{i_j}.

With the encoding strategy defined above, the population is then initialized by randomly scattering the particles into the whole search space. Furthermore, to evaluate the solution quality of a particle p_i during the PSO search, the corresponding fitness function can be expressed as:

$$F_n(S_i) = \alpha \cdot C_v + \beta \cdot E_l \quad (12)$$

where α and β are two weighing factors, C_v and E_l are the numbers of control valves and control paths in the control architecture expressed by p_i, respectively.

Formula (12) implies that the particle positions with fewer control valves and paths will be selected as potential solutions and used to guide the searching of the whole population. The encoding of a particle, however, only indicates the control patterns applied to each multiplexing combination. In order to derive the values of C_v and E_l, the underlying architecture expressed by the particle needs to be determined. Accordingly, we present a control-architecture synthesis approach based on the merging and canceling operations discussed in Section III, which consists of the following two steps: logic reduction for flow valves and logic forest construction.

(1) *Logic reduction for flow valves*: The major goal of this step is to reduce the complexity of the control architecture expressed by a particle, by simplifying the internal logic connected from core input to each flow valve. We take a part of multiplexing matrix of a biochemical application (see below) as an example to illustrate this process.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{matrix} x_1x_2x_3 \\ x_1x_2\overline{x_3} \\ x_1\overline{x_2}x_3 \\ x_1\overline{x_2}\overline{x_3} \end{matrix}$$
$$f_1 \ f_2 \ f_3 \ f_4 \ f_5$$

In this matrix, five flow valves ($f_1 - f_5$) are connected to the core input and a total of four multiplexing combinations are selected to realize the multi-channel control, i.e., $\sum_{i=0}^{2^N-2} l_i = 4, N = 5$. Moreover, the control pattern assigned to each combination in a particle is shown beside each row of the matrix. Fig. 9(a) shows the corresponding logic trees constructed for f_1 to f_5, where root of each tree is the core input, each internal node represents a control valve connected to the corresponding control port, each edge represents a potential control path connecting two valves, and leaf nodes are flow valves. It is then the task to simplify each logic tree by iteratively merging and canceling those redundant control valves. For example, in the logic tree of flow valve f_2, the valve pairs of x_1 and x_2 can be merged, respectively, and both x_3 and $\overline{x}_3$ can be cancelled since they always lead to a connected path to f_2. Similarly, as shown in Fig. 9(b), the logic trees of f_3 and f_5 can also be simplified.

The reduction of a logic tree, on the other hand, is actually equivalent to the simplification of the Boolean logic of the corresponding flow valve, i.e., finding the essential prime

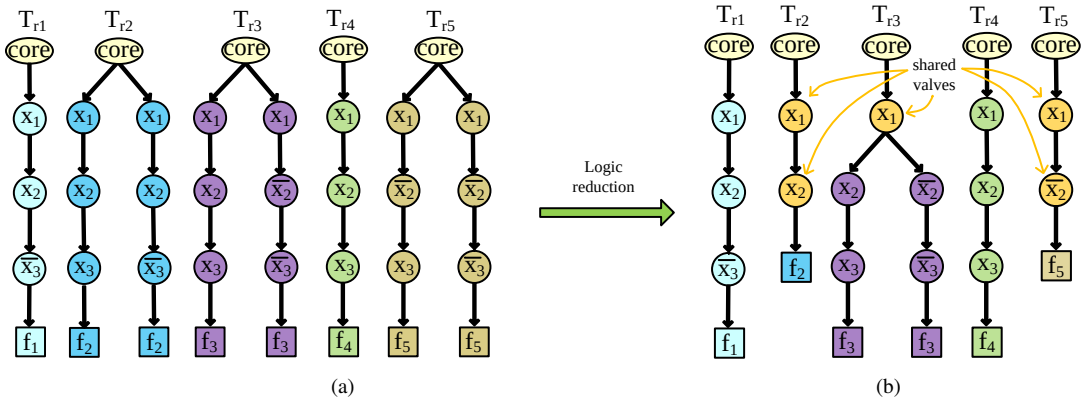


Fig. 9: Logic reduction for flow valves. (a) The initial logic trees constructed for flow valves. (b) Logic trees after internal simplification.

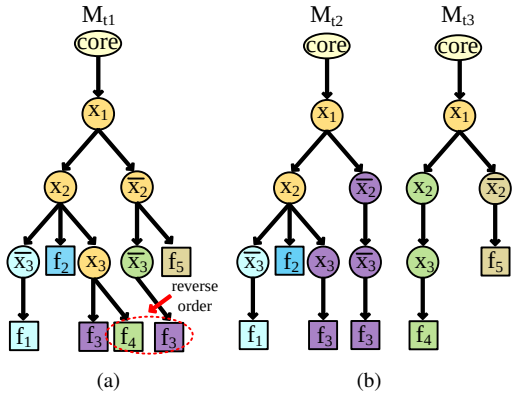


Fig. 10: Logic forest construction for the trees described in Fig. 9(b). (a) M_{t1}=T_{r1}+T_{r2}+T_{r3}+T_{r4}+T_{r5}. (b) M_{t2}=T_{r1}+T_{r2}+T_{r3} and M_{t3}=T_{r4}+T_{r5}.

implicants of its Boolean expression. This problem, however, has been proved to be NP-complete [28]. To identify the redundant control valves so that the internal logic of each flow valve can be simplified in an efficient manner, we adopt the Quine-McCluskey method [29] to realize the merging and canceling operations discussed above. It can be seen from Fig. 9 that the number of control valves in the Boolean logics connected from core input to $f_1 - f_5$ are reduced by 0,4,1,0,4, respectively. In other words, the total number of control valves used in the whole control logic is reduced by 9 through the internal simplification for flow valves. Moreover, the number of potential control paths in the logic (i.e., the number of edges in the logic trees) is also reduced from 32 to 21.

(2) *Logic forest construction*: With the simplified Boolean logic of each flow valve, in this step, the control architecture is further optimized in a global manner by merging the logic trees among flow valves, thereby generating a forest representing the underlying architecture of the final control logic.

We still take the multiplexing matrix mentioned above as an example to illustrate this optimization technique. As shown in Fig. 10, after generating the logic trees of flow valves, these trees can then be merged sequentially to share more control valves and paths, thus further reducing the cost of the whole control logic. Fig. 10(a) shows the resulting logic after merging all the trees in Fig. 9(b), where the logic forest is degraded into a large logic tree in this case. Although the numbers of control valves and paths are reduced to only 6 and 12, respectively, this logic suffers from the following drawbacks: 1) it does not take potential manufacturing defects into account and this, thereby, will lead to plenty of duplications of both control valves and paths when performing the fault-tolerance design and 2) leaf node f_4 appears on the left-hand side of f_3 when merging the logic tree T_{r4} into M_{t1}, leading to a ***reverse-order pair*** of flow valves. When the states of flow valves f_3 and f_5 need to be updated simultaneously, control pattern $x_1\overline{x}_2\overline{x}_3$ will be activated. Since the control path between core input and flow valve f_3 can pass through flow valve f_4, the state of f_4 will be updated at the same time, leading to a logic error. Accordingly, to achieve a tradeoff between the cost of control logic and the efficiency of fault tolerance, meanwhile ensuring the logical correctness of system, we present the following rules to effectively guide the merging of logic trees:

- We set an allowable step size M_s to indicate the maximum number of logic trees that can be merged continuously.
- If reverse-order pairs are introduced into the logic when merging a logic tree T_{ri}, the current merging process will be terminated and a fresh round of merging will be started with T_{ri} as the first logic tree.

Fig. 10(b) shows the generated logic forest when parameter M_s is set to 3, which includes two subtrees M_{t2} and M_{t3}. Compared with the control architecture described in Fig. 9(b), the number of control valves used in the logic is further reduced from 15 to 10, and the number of control paths is reduced from 21 to 16, while the potential for fault-tolerance is also maintained in the control architecture.

After performing the two reduction methods discussed above, the fitness value of a particle p_i, i.e., $Fn(S_i)$ defined in (12), can be derived directly by counting the number of internal nodes and edges in the logic forest, respectively.

Updating Strategy of Particle Population: As illustrated in Fig. 8, after initializing the particle population, the PSO-search is then performed by iteratively updating the position

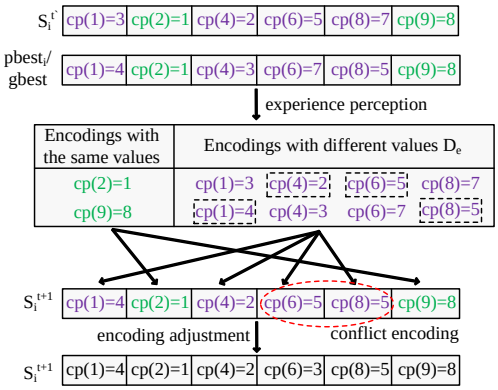


Fig. 11: Experience perception of particles during the PSO search.

of each particle, and thus exploring the whole solution space. Correspondingly, the updating strategy of particles used in the proposed algorithm can be stated as follows:

• The position updating of a particle p_i corresponding to the flying velocity in the t-th iteration can be formulated as

$$S_i^{t'} = U_v(S_i^t, w, r, C_u) = \begin{cases} swap(x_1, x_2), & r < w \\ rep(x_1), & r \geq w \& C_u \neq \emptyset \\ S_i^t, & otherwise \end{cases} \quad (13)$$

where w is the inertia weight in PSO, r is a randomly generated number distributed on the interval [0, 1), C_u is a set of control patterns that has not been assigned to any control channel, $swap(x_1, x_2)$ is an updating function used to exchange the values between two randomly generated encoding positions x_1 and x_2, and $rep(x_1)$ replaces the current value at position x_1 by a new control pattern in C_u.

• The position updating of p_i corresponding to the experience perception in the t-th iteration can be formulated as

$$S_i^{t+1} = U_p(S_i^{t'}, c, r_1) = \begin{cases} I_p(pbest_i), & r_1 < c \\ G_p(gbest), & r_1 \geq c \end{cases} \quad (14)$$

where c is the acceleration coefficient, r is a randomly generated number distributed on the interval [0, 1), and $I_p(\cdot)$/$G_p(\cdot)$ is the individual/global perception function used to exchange encoding information between $S_i^{t'}$ and $pbest_i$/$gbest$. The perception functions are executed through the merging between particle encodings, which consists of the following steps:

(1) Scan the encoding in each parent particle, and select the same encodings as the values of corresponding encoding positions in the new particle.

(2) Copy all the different encodings of two parent particles to a set D_e.

(3) Randomly select encodings from D_e and apply them to the new particle until the values of all encoding positions are determined.

(4) If the newly generated particle is illegal, i.e., the values of some encoding positions violate the constraint (11), replace these conflict encodings by the unused control patterns in C_u.

We take the particles shown in Fig. 11 as an example to illustrate above procedure. In this example, the multiplexing combinations l_i, $i = 1, 2, 4, 6, 8, 9$ are selected to realize the multi-channel control. In the beginning, encodings with the same values in $S_i^{t'}$ and $pbest_i$/$gbest$, i.e., $cp(2)$ and $cp(9)$, are copied to the new particle S_i^{t+1} directly. Furthermore, the remaining encodings of two parent particles, including $cp(1)$, $cp(4)$, $cp(6)$, and $cp(8)$, are added to the set D_e. The encodings in the rectilinear bounding boxes with dashed lines are then selected and applied to the corresponding positions of S_i^{t+1}. Note that since the same control pattern is assigned to $cp(6)$ and $cp(8)$ in the new particle, it is thus adjusted by replacing $cp(6)$ with a new control pattern.

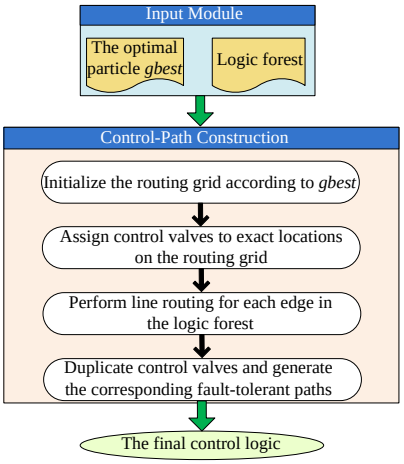


Fig. 12: Flow chart of the proposed control-path construction method.

2) Control-path construction

After a certain number of iterations, the global best solution of the population, i.e., the control architecture expressed by the particle with minimum fitness value, is selected as the underlying architecture to construct the final control logic in the physical design step.

Fig. 12 shows the flow chart of the proposed control-path construction method. In this method, a general routing grid as shown in Fig. 13 is used as virtual guide to construct control paths. 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.

With the logic forest constructed previously, in this step, the control logic is generated by constructing the corresponding paths from core input to each flow valve on the virtual grid. The size of grid is initialized to $C_p \times (L_v - 1)$, where C_p is the number of control ports used in the logic and L_v is set to the total number of leaf nodes in the logic forest. Then the core input as well as the valves used in the control logic can be assigned to exact locations on the grid according to the relative positions of internal nodes in the logic forest. Fig. 13 shows the routing grid as well as the assigned valve locations corresponding to the logic forest described in Fig. 10(b). Since each edge in the logic forest corresponds to a routing net connecting two valves, it is then the task to find control paths such that all the routing nets can be routed successfully. In the proposed method, we traverse each tree in the logic forest and find 'L' routing paths for each net using the well-known line-

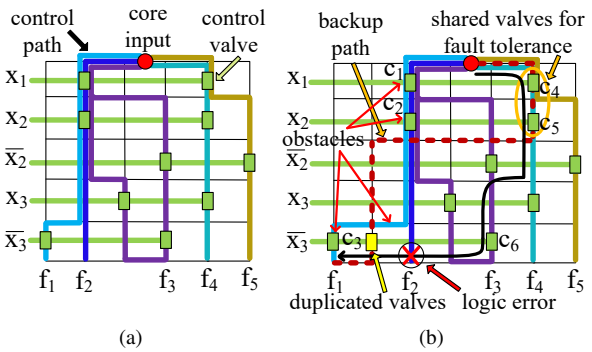


Fig. 13: Physical design of control logic. (a) The constructed control logic corresponding to the logic forest described in Fig. 10(b). (b) Fault-tolerance design for flow valve f_1.

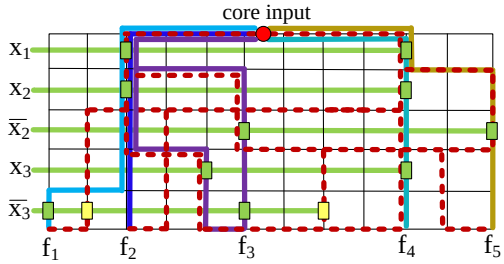


Fig. 14: The final control logic supporting fault tolerance corresponding to the logic forest described in Fig. 10(b).

routing algorithm [30]. Note that there may exist several paths between two control valves on the routing grid. To reduce the total channel length, in other words the cost of control logic, our algorithm tends to share routing paths among different nets, i.e., using the edges occupied by previously routed nets on the grid as much as possible. The constructed control paths corresponding to the routing nets in Fig. 10(b) are also shown in Fig. 13(a), where line segments in different colors represent control paths connected to different flow valves.

After generating all the control paths of the logic system, fault-tolerance design can be performed by computing a back-up path for each flow valve. Since manufacturing defects can result in either the failure of control valves or the blockage of control paths, the original control channel as well as the control valves along the path should be seen as obstacles when finding backup paths for a flow valve. As shown in Fig. 13(b), for example, as we computing the backup path for flow valve f_1, the control path from core input to f_1 and control valves c_1, c_2, and c_3 are seen as obstacles. In other words, any backup path of f_1 cannot reuse these resources. Moreover, valves c_4 and c_5 along the control paths of f_4 and f_5 can be shared to reduce extra cost. Note that valve c_6 cannot be shared since the new path will inevitably pass through f_2, and thereby may lead to a logic error if f_2 is required to be switched independently in the biochemical application. Correspondingly, a duplicated control valve connecting $\overline{x}_3$ is added to the grid. The dashed lines in Fig. 13(b) shows the final backup path constructed for f_1. Besides control valves c_4 and c_5, a part of the control path of f_4 is also shared in the backup path. Similarly, the backup paths of other flow valves can be constructed to generate the final control logic as shown in Fig. 14, in which only two extra control valves are added to implement fault tolerance in the logic system.

TABLE I: Details of benchmarks used in this paper.

Benchmarks (#M_x,#F_v,#C_s,#S_p,#I_p)	
RA30: (2,19,10221,13408,86)	R0: (1,22,5000,6684,153)
mRNA: (3,37,5361,1403,52)	R1: (2,27,6000,8013,244)
CPA: (3,25,2941,1409,92)	R2: (3,48,127000,9372,325)

V. Simulation Results

The proposed method was implemented in C/C++ and tested on a PC with 2.4 GHz CPU and 32GB memory. We demonstrate the results of three real-life biochemical applications that are CPA (Colorimetric Protein Assay) used in RA30 chip from [9], IVD (Int-Vitro Diagnostics) applied in CPA chip from [9], and mRNA chip from [31]. 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 details of aforementioned benchmarks are listed in Table I, where #M_x is the numbers of mixers used in the applications and #F_v is the number of flow valves/control channels. The numbers of states of flow valves in executing the corresponding applications are reported in #C_s and the numbers of switching patterns corresponding to the rows of the switching matrix $\tilde{Y}$ in (2) are reported in #S_p. After merging equivalent rows of switching patterns as described in Section IV-A, the numbers of independent patterns used in (8)–(9) are reported in #I_p. The parameters used in this paper are set as follows: $w = 0.5$, $c = 0.5$, $\alpha = 0.7$, $\beta = 0.3$, and $M_s = 3$. In our simulations, the control logic itself has two layers to implement the multiplexing function. In the flow part, the flow channels and control channels also form a two-layer structure. Therefore, the whole chip can be considered as two two-layer blocks interfaced by the control outputs.

A. Verification of mixing-multiplexing control architecture

In Section IV-A, we presented a mixing-multiplexing control architecture (MMCA) by switching the states of flow valves in mixers in a centralized manner, and thus further improve the efficiency of a control logic. To verify the effectiveness of mixing multiplexing, we compare the design results between MMCA with the architecture in which mixers are controlled individually (IMCA). Fig. 15 shows the comparison results on the total number of valve-switching times. Overall MMCA achieves a 25%–39% switching times reduction across all the benchmarks, with an average reduction of 34%. This significant reduction of switching times will further improve the execution efficiency of biochips. Moreover, as shown in Fig. 16, compared with IMCA, the total number of applied control patterns is also reduced by 10%–29% in the benchmarks, with an average reduction of 21%. This result implies that MMCA has a greater potential to realize a large-scale multi-channel control.

B. Validation of multi-channel switching mechanism

As discussed previously, in the traditional single-channel switching mode, the '1's in a switching matrix must be updated

TABLE II: Validation of the proposed multi-channel switching mechanism.

Bench	Control architecture with individual mixing							Control architecture with mixing multiplexing						
	$\#T_s$	$\#T_m$	Imp (%)	$\#N_c$	$\#A_p$	$\#C_p$	Time (s)	$\#T_s$	$\#T_m$	Imp (%)	$\#N_c$	$\#A_p$	$\#C_p$	Time (s)
RA30	27025	15247	43.6	19	17	10	2.0	27025	10080	62.7	19	12	11	3.3
CPA	4198	1742	58.5	25	22	10	12.3	4198	1065	74.6	25	13	11	1.2
mRNA	4464	1597	64.2	37	20	12	13.5	4464	1055	76.4	37	18	13	38.5
R0	6891	6799	1.3	22	26	10	17.6	6891	5090	26.1	22	20	13	5.7
R1	14334	9776	31.8	27	28	10	37.1	14334	6179	56.9	27	24	13	17.2
R2	26058	11781	54.8	48	51	12	414.8	26058	7481	71.3	48	46	15	275.0
Average	—	—	42.4	—	—	—	—	—	—	61.3	—	—	—	—

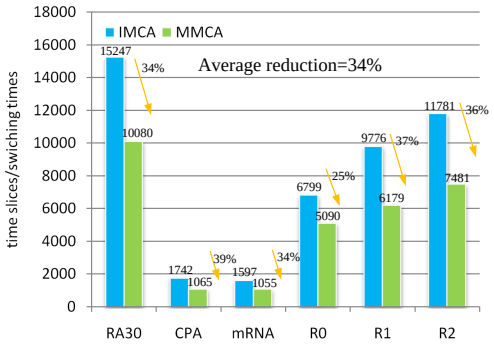


Fig. 15: Comparison on the number of valve-switching times.

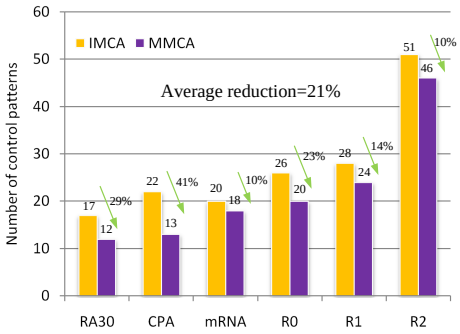


Fig. 16: Comparison on the number of applied control patterns.

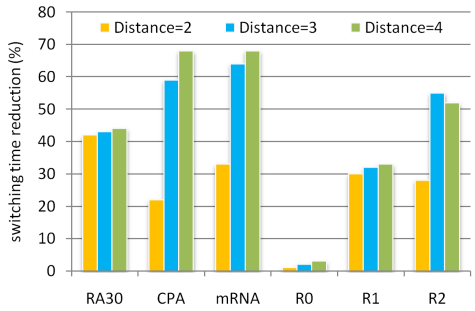


Fig. 17: Reduction of total channel switching times under different multiplexing distances.

individually, leading to a low-efficiency control system. To validate the efficiency of the proposed multi-channel switching mechanism, we compare the synthesis results of two switching modes in both IMCA and MMCA in Table II.

In IMCA, the total numbers of time slices in the single-channel switching mode are reported in the column $\#T_s$ in Table II. With multiple-channel switching, these numbers are reduced significantly in most cases, as shown in the column $\#T_m$. The reduction of these switching times can reach up to 64.2%, as shown in the column Imp.

The numbers of control patterns used in the control logic are shown in the column $\#A_p$, which are larger than the numbers of control channels in the column $\#N_c$ due to the additional control patterns for multi-channel switching for cases R0, R1 and R2, 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 times (42.4% on average) from $\#T_s$ to $\#T_m$ can be achieved. Moreover, the number of control ports used in the control logic are reported in the column $\#C_p$.

In MMCA, compared with the single-channel switching mode, the proposed multi-channel switching mechanism achieves a 26.1%–76.4% time-slice reduction, with an average reduction of 61.3%. Furthermore, in all benchmarks, the numbers of control patterns used in the control logic are fewer than the numbers of control channels, this is achieved by a slight increase of the number of control ports. This result, on one hand, demonstrates the high efficiency of our multi-channel switching scheme, and meanwhile further confirms the effectiveness of the proposed MMCA.

The CPU time to synthesize the control logic by the proposed method is reported in the columns $Time$. It can be seen that all results can be generated within a reasonable time.

In addition, 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 of valve-switching times with different bounds is shown in Fig. 17. As expected, the reduction increases as the number of channels that can be switched together increases. However, a further increase from 3 to 4 does not lead to significant performance improvement, justifying the bound set in our method. For case R2 the reduction even decreases slightly due to the heuristics introduced in the proposed formulation to improve computing efficiency.

C. Validation of control logic construction

In the proposed heuristic for control-logic construction, as we compute the fitness value of a particle, two logic simplification strategies based on merging and canceling operations, i.e., logic reduction for flow valves (internal simplification) and logic forest construction, are proposed to eliminate those redundant control valves. In this part, we first validate the effectiveness of the two methods by comparing the numbers of control valves in each phase. Fig. 18 shows the comparison results without considering fault tolerance. It can be seen that the internal simplification achieves a 13%–33% control-valve reduction, with an average reduction of 22%. On the

TABLE III: Comparison on the cost of control logic.

Bench	Control architecture with individual mixing								Control architecture with mixing multiplexing							
	ILP [1]		Our method		Imp (%)		Time (s)		ILP [1]		Our method		Imp (%)		Time (s)	
	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	ILP [1]	Ours	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	$\#C_v$	$\#C_l$	ILP [1]	Ours
RA30	137	443	135	386	1.4	12.9	1307.0	0.1	75	323	67	209	10.7	35.3	559.5	0.1
CPA	179	629	159	431	11.2	31.5	2146.7	0.1	104	464	98	301	5.8	35.1	846.3	0.0
mRNA	212	949	202	738	4.7	22.2	3159.5	0.1	213	820	205	523	3.8	36.2	2303.6	0.0
R0	274	948	242	691	11.7	27.1	3981.4	0.2	172	740	156	616	9.3	16.8	2465.5	0.1
R1	330	942	218	584	33.9	38.0	4002.9	0.1	332	1110	217	628	34.6	43.4	3942.3	0.1
R2	812	2292	503	1278	38.1	44.2	10171.2	0.2	1170	2669	766	1476	34.5	44.7	9541.8	0.3
Average	—	—	—	—	16.8	29.3	3.1×10^4X	1.0	—	—	—	—	16.5	35.3	3.3×10^4X	1.0

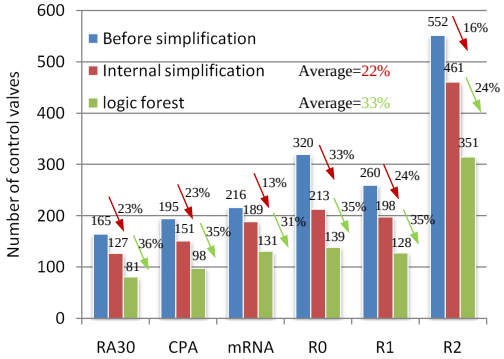


Fig. 18: Validation of the proposed logic reduction methods.

other hand, through the construction of logic forest, control valves that can be merged together among the paths connected to different flow valves can be identified efficiently, thereby leading to a 33% control-valve reduction on average.

Moreover, we compare the proposed heuristic with the ILP method in [1] in terms of the cost of the final control logic, including the number of control valves and the total channel length. Table III shows the comparison results. In IMCA, the proposed heuristic achieves a 1.4%–38.1% control-valve reduction and a 12.9%–44.2% channel-length reduction across all benchmarks, and the average reduction rates reach up to 16.8% and 29.3%, respectively. In MMCA, the number of control valves and the total channel length can also be reduced by 16.5% and 35.3% on average. In other words, our heuristic notably outperforms the ILP method in [1].

The runtimes of the two methods are also reported in Table III. Since the ILP method suffers from the drawback of low efficiency, our heuristic runs much faster than [1] in the benchmarks. The last row of Table III with respect to the CPU time is normalized to our heuristic. It can be seen that our method achieves 3×10^4 times speedup on average in both IMCA and MMCA. In other words, the proposed method is more practical for the microfluidic logic design.

VI. Conclusion

We have studied the control-logic design problem considering both control multiplexing and fault-tolerance in flow-based microfluidic biochips and presented a systematic method to efficiently solve it. By introducing the multi-channel switching mechanism, the time slices required for switching valves can be reduced significantly. Moreover, independent backup paths have also been introduced to improve the reliability of automatically generated control logic. With these concepts, a synthesis framework based on Particle Swarm Optimization, Boolean logic simplification, grid routing, together with mixing multiplexing has been presented. Simulation results have shown that our method can generate a control logic with high efficiency, low cost, and fault tolerance within a short time.

Acknowledgment

The work of Ying Zhu, Bing Li and Ulf Schlichtmann was supported in part by Deutsche Forschungsgemeinschaft (DFG) through TUM International Graduate School of Science and Engineering (IGSSE). The work of Ulf Schlichtmann was also supported in part by 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] Y. Zhu, B. Li, T.-Y. Ho, Q. Wang, H. Yao, R. Wille, and U. Schlichtmann, "Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips," in *Proc. Int. Conf. Comput.-Aided Des.*, 2018, pp. 123:1–8.

[2] K. Hu, K. Chakrabarty, and T.-Y. Ho, *Computer-Aided Design of Microfluidic Very Large Scale Integration (mVLSI) Biochips*. Springer, 2017.

[3] J. M. Perkel, "Microfluidics: Bringing new things to life science," *Science*, vol. 322, no. 5903, pp. 975–977, 2008.

[4] X. Huang, T.-Y. Ho, W. Guo, B. Li, and U. Schlichtmann, "Minicontrol: Synthesis of continuous-flow microfluidics with strictly constrained control ports," in *Proc. Design Autom. Conf.*, 2019, pp. 145:1–6.

[5] 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.

[6] T.-M. Tseng, B. Li, U. Schlichtmann, and T.-Y. Ho, "Storage and caching: Synthesis of flow-based microfluidic biochips," *IEEE Design & Test*, vol. 32, no. 6, pp. 69–75, 2015.

[7] 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.

[8] M. Li, T. Tseng, B. Li, T. Ho, and U. Schlichtmann, "Sieve-valve-aware synthesis of flow-based microfluidic biochips considering specific biological execution limitations," in *Proc. Design, Autom., and Test Europe Conf.*, 2016, pp. 624–629.

[9] 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.

[10] T. Tseng, M. Li, D. N. Freitas, T. McAuley, B. Li, T. Ho, I. E. Araci, and U. Schlichtmann, "Columba 2.0: A co-layout synthesis tool for continuous-flow microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 8, pp. 1588–1601, 2018.

[11] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 6, pp. 1157–1170, 2018.

[12] Z. Chen, X. Huang, W. Guo, B. Li, T.-Y. Ho, and U. Schlichtmann, "Physical synthesis of flow-based microfluidic biochips considering distributed channel storage," in *Proc. Design, Autom., and Test Europe Conf.*, 2019, pp. 1525–1530.

[13] 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.

[14] X. Huang, T.-Y. Ho, K. Chakrabarty, and W. Guo, "Timing-driven flow-channel network construction for continuous-flow microfluidic biochips," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2019, doi:10.1109/TCAD.2019.2912936.

[15] 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.

[16] 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.
[17] 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.
[18] 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.
[19] L. M. Fidalgo and S. J. Maerkl, "A software-programmable microfluidic device for automated biology," *Lab Chip*, vol. 11, pp. 1612–1619, 2011.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] R. Eberhart and J. Kennedy, "Particle swarm optimization," in *Proc. of IJCNN*, vol. 4, 1995, pp. 1942–1948.
[26] X. Huang, G. Liu, W. Guo, Y. Niu, and G. Chen, "Obstacle-avoiding algorithm in x-architecture based on discrete particle swarm optimization for vlsi design," *ACM Trans. Design Auto.Elect. Syst.*, vol. 20, no. 2, pp. 1–28, 2015.
[27] G. Liu, X. Huang, W. Guo, Y. Niu, and G. Chen, "Multilayer obstacle-avoiding x-architecture steiner minimal tree construction based on particle swarm optimization," *IEEE Trans. on Cyber.*, vol. 45, no. 5, pp. 1003–1016, 2014.
[28] T. K. Jain, D. S. Kushwaha, and A. K. Misra, "Optimization of the quine-mccluskey method for the minimization of the boolean expressions," in *Proc. of ICAS*, 2008, pp. 165–168.
[29] W. V. Quine, "The problem of simplifying truth functions," *The American mathematical monthly*, vol. 59, no. 8, pp. 521–531, 1952.
[30] K. Mikami, "A computer program for optimal routing of printed circuit connectors," *IFIPS Proc.*, 1968.
[31] 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.

Ying Zhu received the B.S. degree in optoelectronic information engineering from Huazhong University of Science and Technology, Wuhan, China, in 2013, and the M.S. degree in communications engineering from Technical University of Munich Munich, Germany in 2016.

She is currently pursuing the Ph.D degree with the Chair of Electronic Design Automation, Technical University of Munich. Her research interest is the design automation for emerging computing system.

Xing Huang received the B.S. degree in computer science and technology and the Ph.D. degree in electronic science and technology from Fuzhou University, Fuzhou, China, in 2013 and 2018, respectively. He was a joint Ph.D. with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA, supported by the Chinese Scholarship Council.

He is currently a Postdoctoral Research Fellow with the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan. His current research interests include design automation for microfluidic biochips and integrated circuits.

Bing Li received the Dr.-Ing. degree from Technical University of Munich (TUM), Munich, Germany, in 2010 and finished the Habilitation there in 2018. He is currently a researcher with the Chair of Electronic Design Automation, TUM. His current research interests include high-performance and lower-power design, design automation for microfluidic biochips, as well as emerging systems.

Tsung-Yi Ho (M'08–SM'12) received his Ph.D. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2005.

He is a Professor with the Department of Computer Science of National Tsing Hua University, Hsinchu, Taiwan. His research interests include design automation and test for microfluidic biochips and nanometer integrated circuits.

Dr. Ho was a recipient of the Best Paper Awards at the VLSI Test Symposium (VTS) in 2013 and IEEE TCAD in 2015. Currently he serves as an ACM Distinguished Speaker, a Distinguished Lecturer of the IEEE Circuits and Systems Society, and Associate Editor of the ACM JETC, ACM TODAES, ACM TECS, and IEEE TVLSI, and the Technical Program Committees of major conferences, including DAC, ICCAD, DATE, ASP-DAC, ISPD, etc.

Qin Wang received the B.S. degree in software engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2013, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2018. His current research interests include design automation for microfluidic biochips.

Hailong Yao (M'09–SM'15) received the B.S. degree in computer science and technology from Tianjin University, Tianjin, China, in 2002, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2007.

From 2007 to 2009, he was a postdoctoral research scholar in the Department of Computer Science and Engineering, University of California at San Diego, La Jolla. He is an associate professor in the Department of Computer Science and Technology in Tsinghua University. His research interests include computer-aided design for microfluidic biochips and very large scale integration (VLSI) physical design.

Dr. Yao received two Best Paper Award Nominations at ICCAD in 2006 and 2008, respectively. He received the ISQED Best Paper Award Nomination in 2011, and received the SASIMI Best Paper Award in 2016.

Robert Wille (M'06–SM'15) is Full Professor at the Johannes Kepler University Linz, Austria. He received the Diploma and Dr.-Ing. degrees in computer science from the University of Bremen, Germany, in 2006 and 2009, respectively. Since then, he worked at the University of Bremen, the German Research Center for Artificial Intelligence (DFKI), the University of Applied Science of Bremen, the University of Potsdam, and the Technical University Dresden. Since 2015, he is working in Linz. His research interests are in the design of circuits and systems for both conventional and emerging technologies. In these areas, he published more than 250 papers in journals and conferences and served in editorial boards and program committees of numerous journals/conferences such as TCAD, ASP-DAC, DAC, DATE, and ICCAD. For his research, he was awarded, e.g., with a Best Paper Award at ICCAD, a DAC Under-40 Innovator Award, a Google Research Award, and more.

Ulf Schlichtmann (M'90–SM'18) received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering and information technology from Technical University of Munich (TUM), Munich, Germany, in 1990 and 1995, respectively.

He is Professor and the Head of the Chair of Electronic Design Automation at TUM. He joined TUM in 2003, following 10 years in industry. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems. Increasingly, he focuses on emerging technologies such as lab-on-chip and photonics.