

Chapter 2

Control-Layer Optimization

In this chapter, we present the first practical problem formulation for automated control-layer design in flow-based mVLSI biochips and propose a systematic approach for solving this problem. Our goal is to find an implementable solution for timing-efficient control-layer design with a minimum number of control pins [1, 2]. To reduce the number of control pins without losing much-needed functionality or operational flexibility, we propose an operation-based compatibility identification method that relies on the compatibilities between valves based on component actions, instead of a pre-determined and complete control-logic table. To increase the likelihood of success in routing, two incremental control-layer design methods are presented. These approaches take a global view of the impact of routability at each design stage by incrementally increasing the routing priorities of failed routing attempts. Furthermore, the impact of pressure-propagation delay, which is a specific physical phenomenon in mVLSI biochips, is minimized in order to reduce the response time of valves, reduce the pattern set-up time, and synchronize valve actuation. Two fabricated flow-based devices and six synthetic benchmarks are used to evaluate the proposed optimization method. Compared with manual control-layer design and a baseline approach, the proposed approaches lead to significant reductions in the number of control pins and channel length in all cases.

The remainder of this chapter is organized as follows. The research motivation is introduced in Sect. 2.1. Section 2.2 describes the design requirements, design challenges, and routability concerns at each design stage, as well as their inter-relationships. Specific constraints due to the physical characteristics of flow-based biochips are also presented. Section 2.3 formulates the optimization problem being studied in this work. Section 2.4 details two proposed algorithm for solving the optimization problem. Finally, two fabricated flow-based devices and six synthetic benchmarks are used to demonstrate the effectiveness of the proposed approach in Sect. 2.5. Conclusions are drawn in Sect. 2.6.

2.1 Motivation and Related Prior Work

The standard design procedure for flow-based microfluidics consists of two stages: flow-layer design and control-layer design. Given a biochemical application and a microfluidic component library, the flow layer of a flow-based microfluidic biochip can be designed in two steps: (i) allocation of components from a given library, (ii) physical synthesis, i.e., deciding the placement of the microfluidic components on the chip and performing routing of the flow channels to create component inter-connections [3, 4]. Although design automation of the flow layer has been studied in considerable detail, automated design and optimization of the control layer has largely been neglected. The objective of control-layer design is to implement the functionality of the flow layer by establishing the connections between valves and control pins through control channels. In prior work, a channel-routing method for flow-based microfluidic biochips was proposed in [5]. However, this approach imposes a fictitious (and completely unnecessary) constraint that control pins must be placed at the edge of the chip. In reality, control pins can be placed anywhere on the chip [6].

2.2 Problem Description, Design Requirements, and Challenges

This section presents the constraints and challenges involved in control-layer design in flow-based mVLSI biochips. First, we highlight a specific characteristic in flow-based mVLSI biochips, i.e., the pressure propagation delay in microchannels, and its impact on our design problem. Next, three design requirements and other challenges for each design stage (valve addressing, control-channel routing, and control-pin placement) are described in a sequential manner.

2.2.1 Pressure-Propagation Delay

Unlike traditional routing problems such as in PCBs and VLSI design, where signal propagation is electronic, the velocity of pressure-signal transportation in control channels is a major limitation in mVLSI chip optimization. When the pressure signal is generated by an external pressure source and injected from a control pin, valves cannot be closed immediately. The pressure-propagation delay in microchannels can be modeled as a ladder RC circuit model [7, 8]. Pressure will dilate microchannels due to the flexibility of biochip materials, i.e., the energy of compressed air will be transferred and stored as mechanical potential energy of the channel walls. This energy storage can be modeled as a hydrodynamic capacitor. Similarly, the flux of air/water flow is restricted by microchannels; microchannels can be modeled as

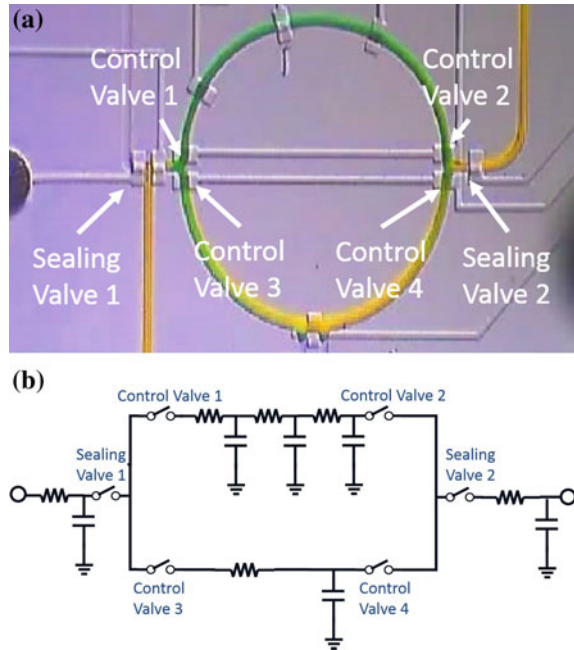
hydrodynamic resistors, whose resistance is linear in l/w^2 (w : cross-section area of a microchannel; l : channel length) [9]. Figure 2.1b illustrates an example of an RC circuit model for the layout shown in Fig. 2.1a. For a given design specification, the hydrodynamic capacitance and cross-section area of control channels are predetermined. Therefore, we can simplify the modeling of propagation delay by making an assumption that the pressure signals propagate with constant velocity, v , in the control channels. A typical value of v is 10 mm/s [10]. The pressure-propagation delay leads to the following problems:

1. Longer response time for valve actuation: As discussed above, valves cannot respond instantaneously to a pressure source due to the pressure-propagation delay in the control channels. The slow response increases the completion time for biochemical experiments.
2. Longer pattern set-up time: To ensure that all valves are completely switched to the desired states, a pattern set-up time has to be inserted between the executions of two consecutive valve-actuation patterns. The preparation time must be larger than the delay caused by the longest pressure-propagation path so that all valves can be completely set to the appropriate states. Because this interruption happens before the execution of each valve-actuation pattern, the pattern set-up time is a key obstacle to shortening the experiment execution time.
3. Asynchronous valve actuation: The pressure signals are injected from control pins and propagated along control channels. Thus, valves sharing the same control pin cannot be opened or closed simultaneously if their distances to the control pin are different. Such asynchronous valve actuation can cause malfunctions. A typical example arises when we have to seal valves at the two ends of a typical mixer (Fig. 2.1). These valves are designed to seal liquid inside the mixer. Once reagents are loaded into the device, the mixer loop is sealed via the sealing valves and the peristaltic pumps are activated sequentially to force the liquid inside to rotate. If sealing valves cannot be closed simultaneously, the closed valve may push liquid out of the mixer. As a result, the reagent cannot be mixed in the desired mixing ratios.

2.2.2 Requirements in Control-Layer Design

In addition to control-channel routing and the timing concerns caused by pressure-propagation delay, there are three important requirements in the design of the control layer. These requirements make control-layer design especially important for handling complexity and reducing cost [11]. The first requirement is the reduction of the number of control pins, which can be realized by connecting a control pin to multiple valves, i.e., *valve addressing*. Although the valves sharing a single control pin can no longer be controlled independently, the minimization of pin count is the primary goal as long as routing in the control layer can be implemented. The minimization of pin count needs to be given the highest priority due to the following reasons:

Fig. 2.1 **a** Layout of a typical mixer. **b** RC circuit model for **a**. Two sealing valves must be closed simultaneously to ensure that the liquid remains in the mixer. Otherwise, liquid may leak from the mixer



1. Pins on the chip increase the difficulty of fabrication and take up additional on-chip area. The access to control pins is provided by a punched hole. To avoid damaging the microchannel network, a “keep out” safety zone, whose radius is approximately 2 mm, must be set around the control pins [12]. Otherwise, the punch may lead to fluid leakage that can damage the chip. The safety zone around a control pin takes up large on-chip area, and therefore jeopardizes the routability of control channels.
2. Each control pin must be connected to an external pressure source. As a result, a large, expensive, and complex controlling system around the chip is required if each valve is directly addressed with an independent control pin [6, 13].

The second requirement is the routability. Minimization of control pins may result in the design being unroutable because each control pin and its corresponding valves must be inter-connected. Similarly, an inappropriate placement of control pins can increase routing complexity because the safety zones around control pins form routing obstacles. Hence, routability concerns should be considered globally and jointly with pin-count minimization to increase the routing success rate.

The third requirement is related to microfluidics-specific timing considerations. Due to the flexibility of the PDMS material, pressure propagation in a PDMS microchannel is very slow [10]. This physical characteristic leads to a number of specific design concerns, such as the response time of valves and asynchronous valve actuation, and it makes our problem considerably different from traditional wire-routing problems in VLSI chips.

Fig. 2.2 Comparison among routing results with different routing targets. **a** Routing after minimization of control-pin count. **b** Routing after optimization for timing. **c** Routing after optimization for routability. **d** Routing after global consideration of pin-count minimization, timing concerns, and routability

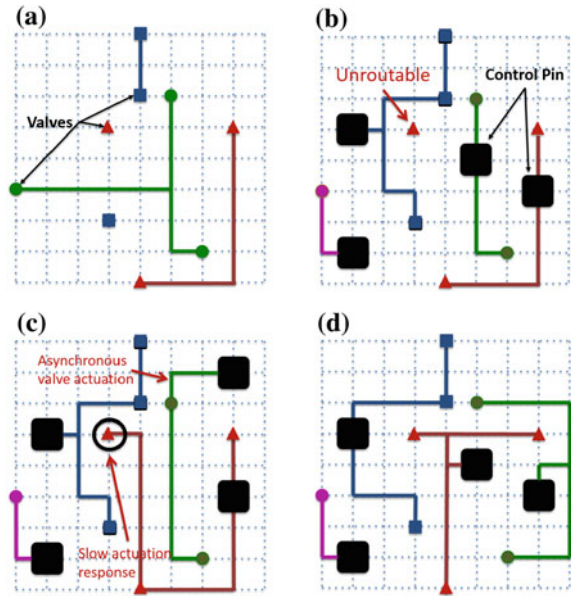


Figure 2.2 illustrates why all the three requirements discussed above must be considered carefully and in a global manner. In Fig. 2.2a, three sets of valves (red triangles, blue rectangles and green circles) are placed in a nine-by-nine grid. Valves in each set can share a single control pin, i.e., at least three control pins are required. However, due to the interconnection of the green set, the minimization of control pins leads to an unroutable solution. Hence, we have to add one additional control pin, i.e., partition the green set into two (the green and violet sets in Fig. 2.2b). We first route the green and blue sets. To shorten the response time, valves are routed with the shortest paths; to synchronize valve actuations, control pins (the black rectangle) are placed either at, or connected to, the center of routing paths. Assume that the shape of a safety zone is a three-by-three rectangle. As a result, a valve in the red set turns out to be unroutable because all possible routing paths are blocked. We can prevent the blockage by moving away the control pin of the green set (Fig. 2.2c). Although the routability is improved, the resulting design introduces two new problems: (1) the green set suffers from the serious problem of asynchronous valve actuation; (2) the actuation response of the red valve in the black circle is extremely slow because its pressure-propagation path is too long. An optimized routing result is showed in Fig. 2.2d. Not only are all valves and control pins successfully routed, the timing behavior of this design is also much better: (1) the valve actuations of each set are synchronized; (2) the actuation response of each valve is acceptable as well.

2.2.3 Valve Addressing

The major goal of valve addressing is to partition the set of valves into multiple groups. All valves in a group share a single control pin and therefore have to be driven simultaneously. Despite the loss of control flexibility, this design stage can reduce the number of control pins. Valve addressing consists of two sub-tasks: compatibility identification and control-pin minimization.

2.2.3.1 Operation-Based Compatibility Identification

Since valves assigned to the same control pin are driven simultaneously, these valves must be mutually “compatible”, i.e., the same control pin can drive them. In this subsection, we attempt to reduce the number of control pins by identifying the compatibility among valves. Compatible valves can be assigned to the same control pin without any control conflicts.

In prior work [13], the determination of compatibility is carried out using a control-logic table, called table-based compatibility identification (TBCI), in which the activation sequences of all valves at every time frame are explicitly listed. In addition to the value “1” (valve open) and value “0” (valve closed), there are a number of “X” s in the table to denote “don’t-care” values; the corresponding valve can be set to an arbitrary state without interfering with fluid operations at that time. By replacing these “X” values with either 1s or 0s, the activation sequences of certain valves can be made identical and therefore they are mutually compatible. Note that the compatible valves assigned to the same control pin cannot be controlled independently. As a result, the correctness and completeness of the control-logic table is crucial for the effectiveness and flexibility of chip design. Only the valve-activation patterns included in the control-logic table are guaranteed to be executable, while fluid-handling operations that are not included in this table may no longer be supported. However, a control-logic table is generated by mapping experiment plans to activation sequences of valves. Due to the inherent randomness and complexity of the component interactions that is ubiquitous in biochemistry, the experimental plan cannot be predicted accurately and may need to be modified adaptively based on intermediate reaction results or a change in the environment. In other words, the control-logic table cannot be pre-determined at the design stage of a biochip.

For example, Fig. 2.3a shows the layout of an mVLSI cell-culture biochip with a 2-1 multiplexer (the branch), a mixer (the loop), and two culture chambers. The cell culture can be executed on this chip by: (1) mixing samples and culture media in the mixer; (2) loading the mixture into a chamber. However, the experimental results of cell culture are unpredictable; the outcome depends on a large number of experimental conditions and parameters, such as the cell-feeding volumes and schedules, the concentrations and types of culture media, the mixing cycles and frequencies, etc [14]. Therefore, to improve the culture quality, culture conditions and the experiment plan needs to be modified adaptively during the experiment. As a

result, it is not possible to generate the control-logic table without simply enumerating all possible flow paths. Furthermore, since the number of possible combinations of valve states increases exponentially with the growth of chip size and complexity, it is difficult to exhaustively list all necessary activation patterns of valves in a large-scale flow-based biochip. The incompleteness of a control-logic table may lead to unsupported activation patterns of valves, i.e., loss of functionality, and makes the chip design too inflexible to be practical.

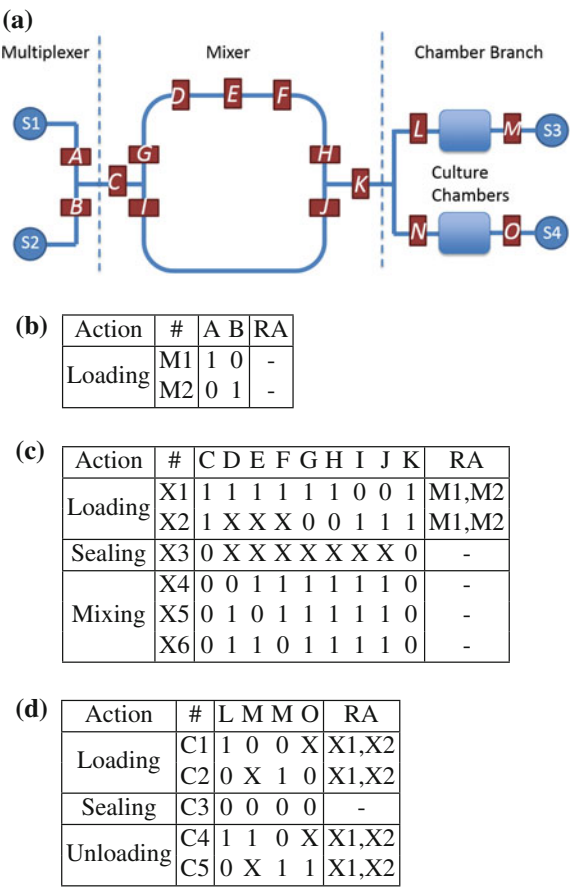
Let us revisit Fig. 2.3 as an example. To increase throughput and provide greater flow control, we should: (1) increase the size of the multiplexer to provide access to different culture media and other reagents, e.g., surface treatment reagents, staining reagents, and flushing reagents; (2) increase the number of mixers to support different mixing ratio; (3) increase the number of chambers for more control groups and higher throughput. Therefore, the total number of activation patterns grows as $O(mnl)$, where m is the branch count of the multiplexer, n is the number of mixers, and l is the number of culture chambers. (A cell culture chip fabricated at the Stanford Foundry in 2007 [14] has 16 inputs ($m = 16$) and 96 culture chambers ($n = 96$).) The rapid increase in the number of possible valve-activation patterns clearly makes the generation of a complete control-logic table infeasible. If an activation pattern of valves is not included in the control-logic table, the corresponding modification of culture conditions may be infeasible, and consequently, the flexibility and practicability of the resulting chip design will be significantly reduced.

We propose an operation-based compatibility identification (OBCI) method to identify compatible valves to enhance the operation effectiveness and flexibility of mVLSI biochips. In the synthesis of the flow layer [3, 4], a protocol or application can be scheduled as a sequence of fluidic operations (e.g., mix, dilution, detection). A number of fluidic components, as basic functional modules, are bound to these operations and placed in the flow layer. The execution of a standard component can be divided into several control actions. Thus, instead of the establishment of a complete pre-determined control-logic table, we identify compatibility based on the control actions of each component to avoid loss of functionality. For example, to execute the mix operation in a mixer, whose layout is shown in Fig. 2.3a, three component actions of the mixer can be executed as follows:

1. Mixer loading/unloading: Samples are loaded/unloaded into the upper/lower halves of the mixer;
2. Sealing: Sealing valves (Valve C and Valve K) are closed to contain samples in the mixing loop;
3. Mixing: Three pneumatic valves (D, E, and F) are activated sequentially at a sufficiently high frequency (100 Hz) to force samples to rotate and mix.

The component actions of commonly used components, such as the mixer and multiplexer, can be pre-determined and stored in a library. The allowed control actions of the multiplexer, the mixer, and culture-chamber branch are shown in the Fig. 2.3b–d, respectively. The columns “RA” denote the related component actions (will be discussed later). A control action may consist of one or multiple valve-actuation patterns, such as for the mixing action described earlier. Note that component actions

Fig. 2.3 Layout of a simple mVLSI cell-culturing biochip with a 2-1 multiplexer (the branch), a mixer (the loop) and two culture chambers (the rectangle); **b** Component actions of the multiplexer; **c** Component actions of the mixer; **d** Component actions of the culture-chamber branch



such as sealing, mixing can be executed independently, while the loading actions of a mixer must be coordinated with the multiplexer to define the source for loading the mixer. We use the term *functional action* to refer to a fluid-control action of an entire biochip. For a specific protocol or application, we can simulate the execution of fluid-handling operations on a biochip by describing it as a schedule of functional actions, instead of a complete control-logic table. A functional action can be constructed by expanding the corresponding component actions according to two rules: (i) valves of the components that are not involved in the functional actions are set to “don’t care” values. For example, the functional action of component action X3-X6 are performed only by the mixer; therefore, all valves that belong to the multiplexer and the chamber branches are set to “X” because they are idle for this action; (ii) for component actions that cannot be executed independently, the functional actions can be constructed by taking the Cartesian product of all the related sets of component actions from other components.

Let us take Fig. 2.3 as an example. The “RA” column in Fig. 2.3c shows that the execution of the loading actions of the mixer, i.e., X1 and X2, need to be coordinated with the multiplexer, i.e., M1 and M2. Hence, the functional actions of component actions X1 and X2 can be obtained by the Cartesian product of the set {M1, M2} and {X1, X2}, whose fluid-handling functionality can be described as “loading of Sample S1/S2 into the upper/lower half of the mixer”. Similarly, the functional actions of component actions C1 and C2 can be obtained by the Cartesian product of the sets {C1, C2}, {X1, X2} and {M1, M2}. Note that the set {M1, M2} is inherited because it is related to component actions {X1, X2}. After all functional actions based on the combination of component actions are found, compatibility among valves can be identified without loss of functionality. In summary, despite leading to a slight increase in the number of control pins, the proposed OBCI method offers two advantages over the TBCI method proposed in [13]: (i) OBCI provides full coverage of functional actions, and thus it can provide much-needed experimental flexibility for realistic scenarios; (ii) TBCI requires a pre-determined control-logic table, while OBCI does not require this tedious step. The construction of a control-logic table is infeasible if the experiment need to be adaptively modified in response to intermediate experimental results.

2.2.3.2 Control-Pin Minimization

A compatibility graph can be constructed to represent the compatibility between microvalves [13, 15]. In this graph, each vertex represents a valve and an edge connecting a pair of vertices indicates that they are mutually compatible. Using this compatibility graph, the valve-addressing problem can be mapped to the minimum clique-partitioning problem [16], which has been studied extensively in the literature. A clique is a complete subgraph in which any two vertices are connected by an edge. Therefore, all valves in a clique can be placed in the same group and they can share a single control pin without any control conflicts.

2.2.4 Routing of Control Channels

The goal of control-channel routing is to implement the valve-addressing solution; in other words, the objective is to connect all valves that are partitioned into the same group by control channels. The routing of valves in each clique forms a tree that spans all of the valves in this group. Note that control-pin placement has not yet been determined in this stage. Due to the pressure-propagation delay in PDMS channels, the length of the control channels should be minimized to reduce the response time for valve actuations. We use a maze router based on Lee’s algorithm [17] to find the shortest path between any two valves in the same group. To consider routability globally in each design stage, a routing sequence is created to guide the implementation of the router. In particular, by moving the failed routing pairs forward in the routing

sequence, valve pairs that are difficult to route are routed earlier and can therefore utilize more routing resources. However, if a valve pair fails during multiple routing attempts, it will be considered “unroutable” and will be avoided in subsequent routing iterations. As a result, more control pins may be required in the valve-addressing design stage to achieve an implementable design.

2.2.5 Placement of Control Pins

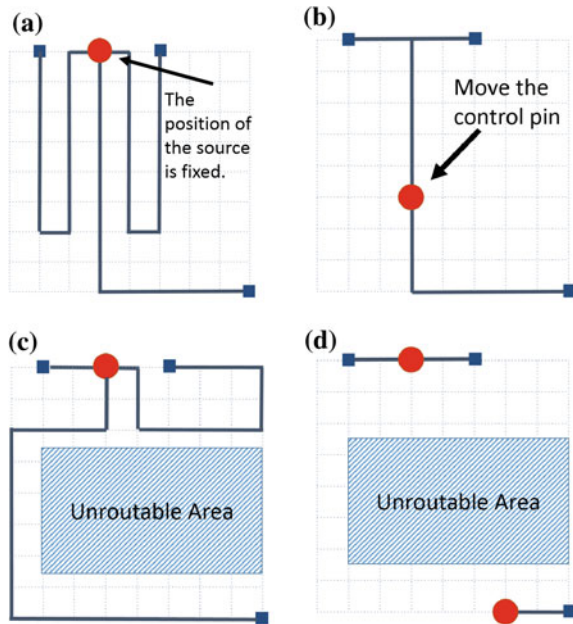
The objective of this design stage is to place control pins on the chip and connect them to the corresponding routing trees (the union of valves and control channels in the same group). It is desirable to place control pins either at, or connected to, the center of the routing trees in order to: (i) balance the routing distance between the valves and control pins and thus alleviate the problem of asynchronous valve actuation; (ii) reduce the pattern set-up time, which depends on the longest pressure-propagation path. To avoid damaging the control channels during fabrication, control pins must be placed a safe distance away from control channels that belong to other groups, and this safety zone around the control pins can occupy a significant amount of on-chip area (4 mm^2) [12]. Consequently, without careful and systematic optimization, it is often difficult or even impossible to determine the placement of control pins due to the lack of available space.

2.2.6 Relationship Between Control-Layer Optimization and Clock-Tree Design in VLSI Circuits

In VLSI circuit design, clock skew refers to the scenario in synchronous circuits in which the clock signal arrives at different sinks (flip-flops) at different times. The objective of clock-tree design is to distribute the clock signals from a single source to all the sinks with negligible skew. Here we discuss the similarities and differences between control-layer optimization and clock-tree design. Although the pressure-propagation delay is analogous to the signal delay through wires in a clock network, the control-layer optimization problem for mVLSI and the clock-tree design problem for VLSI are different in the following aspects:

1. In the clock-tree design problem, the minimization of clock skew is required and must be given top priority. However, not all asynchronous valve actuations will lead to malfunctions. For example, asynchronous valve actuation is acceptable if two mixing operations are executed independently in two mixers. Since there are multiple objectives in the formulated problem, and clearly, they cannot be achieved at the same time in practice, we choose to judiciously accept a small amount of skew for fewer control pins and shorter channel length in some cases. Hence, in the control-layer optimization problem, skew minimization is preferred,

Fig. 2.4 Two examples to illustrate the differences between the clock-tree design problem and the control-layer optimization problem. **a** Because the sources/sinks are fixed in clock-tree design, snake routing is used to balance the skew. **b** Moving the control pin to the center of the tree significantly reduces the pattern set-up time and total channel length. **c** Due to the unroutable area, the latency of the clock tree is large. **d** In control-layer optimization, one additional control pin is added to reduce the pattern set-up time



but it is not the primary goal. The minimization of the number of control pins and pattern set-up time have higher priorities.

2. The clock sources and sinks are fixed in the clock-tree design problem. In contrast, we can move the control pins or add more pins to increase routability and reduce the pattern set-up time. Figure 2.4 describes two examples to illustrate this difference. Circles refer to clock sources in the clock-tree design problem and control pins in the control-layer optimization problem, respectively. Rectangles indicate sinks in the clock-tree design problem and valves in the control-layer optimization problem, respectively.

2.3 Problem Formulation

We next give a formal statement of the problem of optimizing the control-layer design. Because inlets/outlets are fabricated by punching holes through the control layer, no routing in the control layer is allowed at the inlets/outlets of the flow layer.

Input: The valve positions, the locations of the inlets/outlets of the flow layer, component actions of the on-chip fluidic components, functional actions necessary for the targeted protocol, the size of the required safety zone around a control pin, and design rules such as the minimum control channel width and separation.

Output: An implementable routing solution for the control layer, including the routing of control channels and the placement of control pins.

Constraints:

1. All valves assigned to the same control pin must be mutually compatible.
2. All design rules must be satisfied.
3. No routing is permitted in the safety zone around control pins. Moreover, no valve is permitted in the safety zone.
4. No routing is permitted at the inlets/outlets of the flow layer.

Objectives:

1. Minimize the control-pin count to reduce chip size and the cost of the supporting systems.
2. Minimize the channel length to reduce the response time for valve actuation.
3. Minimize the longest pressure-propagation path (latency) to reduce the pattern set-up time.
4. Minimize the channel length imbalance (skew) in each group to avoid asynchronous valve actuation.

Recall that Objectives 2 through 4 can be realized by searching the shortest routing paths and placing a control pin at the center of the routing tree. However, the safety zones around control pins lead to obstacles where control-channel routing is forbidden. Also, minimizing the total pin count (Objective 1) may increase the routing complexity and produce an unroutable design. Hence, routability is another important concern in our problem.

2.4 Algorithm Design

The algorithm for design automation and optimization of the control layer has three main parts: valve addressing, control-channel routing, and placement of control pins. To improve the routing success rate, the utilization of the routing resources in each task should be considered carefully and in a global manner. We first determine the routing priorities of pin-to-valve and valve-to-valve connections, which are then sorted to create the routing sequence. A connection with a higher routing priority will be at the top of the routing sequence, and will therefore be routed earlier in the next iteration. In each iteration, we record any connection that cannot be routed. A routing failure can occur due to one of the following reasons:

1. All routing paths are blocked by previous routing results. In this case, the corresponding valve pairs or valve-pin pair should be routed earlier to avoid congestion (i.e., we have to increase the routing priority of failed routing attempts).
2. The valve pairs are completely unroutable due to an over-aggressive valve-addressing strategy. In this case, the failed routes should be avoided, which can be accomplished by labeling the corresponding valve pairs as “incompatible”.

In this chapter, we propose two control-channel routing algorithms. The first algorithm focuses on the improvement of routability; therefore, it has higher likelihood of finding a routable solution. The method generates a routing sequence by evaluating and sorting the routing difficulties of each valve pair. Although this method significantly increases routability, the routing quality is degraded because the routing process is continuously disrupted by the area that is temporarily reserved for the remaining control pins. The second algorithm focuses on routing quality; therefore, it leads to lower latency and smaller channel lengths. In this algorithm, each valve group is routed and optimized separately and a control pin is not assigned until the routing tree is completely established and optimized. However, the routability of this method is worse than that for Algorithm 1 because fewer routing sources are available for control pin placement in the later stages of the optimization, which increases the likelihood of unroutable results.

2.4.1 Routing Algorithm 1

The flow chart of the proposed Algorithm 1 is presented in Fig. 2.5. We next explain the flowchart step-by-step.

Step 1: All control pins, control channels, and obstacles (inlets/outlets of the flow layer) are mapped to a grid. To simplify our algorithm, the grid size is defined by the minimum center-to-center distance between control channels. Thus, the control channels can be routed through all of the unmapped grid points without violating the design rules.

Step 2: See operation-based compatibility identification (Sect. 2.2.3.1).

Steps 3–4: The goal of these two steps is to initialize the routing priority by estimating congestion and routing cost. The congestion is an evaluation of the routing resources (grid points). The congestion value at each grid point consists of a vertical component and a horizontal component, because only channels in the perpendicular direction may hinder a route. The congestion value of a grid point contributed by a valve pair can be evaluated by considering the likelihood that the grid point is occupied by the routing path between the valve pair. To simplify our problem, we assume that all valve pairs are routed with the simplest and most common path format: a shortest path with at most two bends. Note that this assumption is made only for congestion estimation. In the final routing stage, control channels can be routed in any necessary shape. Figure 2.6 illustrates the procedure of congestion estimation. Several simple paths to connect the valve pairs colored red and blue are shown in

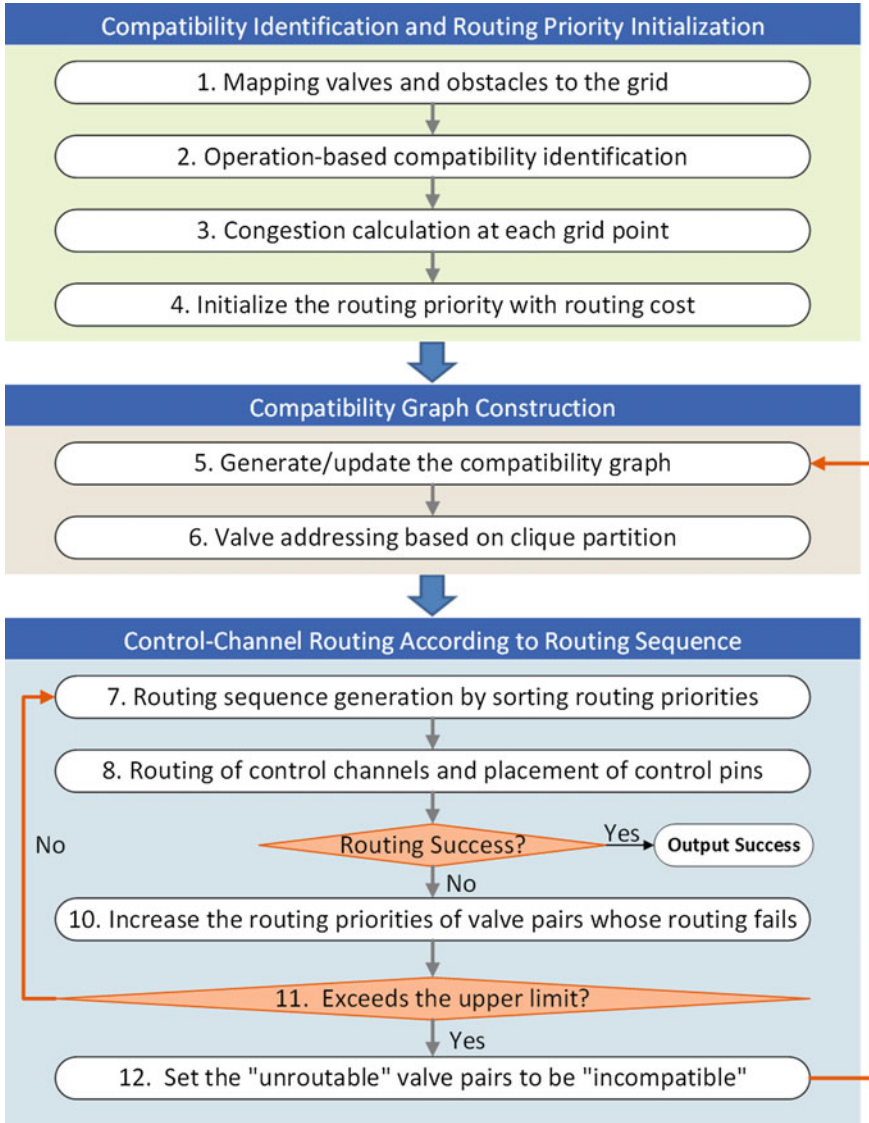
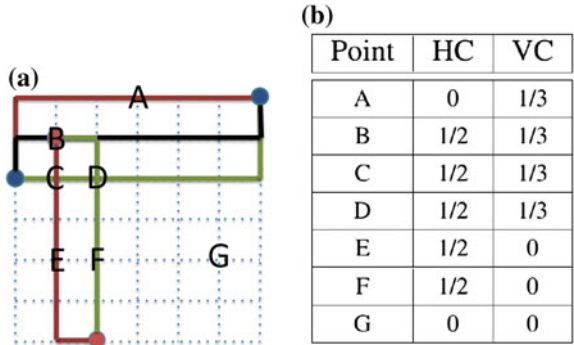


Fig. 2.5 Flowchart of routing Algorithm 1

Fig. 2.6a. Assume that these routing paths have the same likelihood of being chosen in the final routing result. Hence, for the routing of red valve pairs, the likelihood that grid point is vertically passed through is $1/2$ in the bounding box and 0 elsewhere. Similarly, the likelihoods that a grid point is horizontally passed through by the routing of the blue valve pair is $1/3$ in the bounding box and 0 elsewhere. Therefore, the congestion caused by a valve pair can be calculated by $1/W$ (W : width of the

Fig. 2.6 Illustration of the congestion and routing cost calculation. **a** The simplest routing path format between two valve pairs; **b** Congestion calculation results of the points labeled in **a**



bounding box) for horizontal congestion and $1/H$ (H : height of the bounding box) for vertical congestion, while the total congestion of a grid point can be quantified with the sum of congestions caused by all routing valve pairs in the corresponding direction. Figure 2.6b shows the total congestion of the points labeled in Fig. 2.6a. “HC” denotes the horizontal component of congestion, and “VC” denotes the vertical component. The routing cost of a path can be obtained by the sum of the congestion value of all on-path grid points in the corresponding direction. The routing cost of a valve pair can be estimated by the minimal routing cost among all simple paths between them.

Let us revisit Fig. 2.6 as an example. The red paths are chosen for the calculation of routing cost for both valve pairs. The routing cost of the blue valve pair is zero, while the routing cost of red valve pair is $2/3$ due to the vertical congestion components at Point B and Point C caused by the blue valve pair. The estimation of routing cost can improve routing quality by guiding the utilization of routing resources (grid points); valve pairs that require higher routing cost are hard to route and should thus be considered earlier to avoid congested regions. In our algorithm, the routing cost is used to initialize the routing priority in order to improve the routing performance. Based on the the estimation of routing cost and the adjustment of routing priority, the routing path search is accelerated.

Step 5: An undirected graph is constructed to represent compatibility; a vertex in the compatibility graph represents a valve, while an edge indicates that the two vertices connected by this edge are mutually compatible.

Step 6: The goal here is to minimize the number of control pins by partitioning compatible valves into a minimum number of groups. According to Sect. 2.2.3, this problem can be formulated into a minimum clique partitioning problem. Although the minimum clique partitioning problem is NP-hard [16], an exact branch-and-bound algorithm proposed in [18] is efficient for many experimental test cases. This will be demonstrated in Sect. 2.5.

Step 7: A routing sequence is generated so that valve pairs can be routed one by one according to their positions in the routing sequence. The route generation procedure is composed of three steps: First, a set of edges is created by removing all

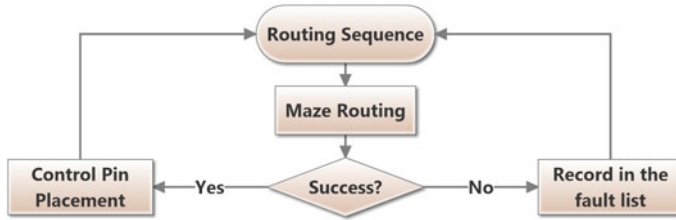


Fig. 2.7 Flowchart of the routing engine (Step 7 in Fig. 2.5)

inter-clique edges. This set of edges is then sorted in ascending order according to their routing priorities. Finally, if both valves covered by an edge have been routed, this edge is considered redundant and therefore removed from the routing sequence.

Step 8: In order to implement the result of valve addressing, valve pairs are routed one by one according to the routing sequence. The flowchart for the routing engine is shown in Fig. 2.7. Valve pairs that cannot be successfully routed are recorded in a fault list. Their routing priorities will be increased later (Step 10 in Fig. 2.5). Recall that the preferred placement of control pins is the center of a routing tree to reduce latency and skew (Objectives 3–4 in Sect. 2.3). However, sometimes sufficient space is not available around the tree’s center because the control pin and its safety zone take up a large amount of routing space. Therefore, the routability of control layer and the timing concerns must be considered globally. We attempt to place a control pin at the center of the current routing tree or at a nearby location after each successful control-channel routing step, even if more valves might be connected to the routing tree later. The effectiveness of this method is based on two realistic assumptions: (1) As the routing tree becomes larger, the position of the tree’s center will not move significantly after each incremental expansion of the routing tree; and (2) Placement of control pins becomes harder as more routing resources are occupied.

Step 9: If all valves in the routing sequence are successfully routed and their control pins are placed, the final routing result will be provided as output. Otherwise, the failed routing attempt will be recorded in a fault list for feedback.

Steps 10–12: The routing priorities of all valve pairs in the fault list will be increased. Moreover, an upper limit will be set to the routing priority. Valve pairs whose routing priorities exceed this limit will be labeled “unroutable”. To avoid any further attempts to route “unroutable” valve pairs, the compatibility graph will be updated by removing edges between them. As a result, valve addressing based on the new compatibility graph will assign unroutable valves into different cliques, though more control pins may be required due to the loss of compatibility. However, if no priority value exceeds the upper limit, the routing sequence will be updated by moving the failed pairs forward. Consequently, more routing resources are available for these valve pairs in the next iteration.

2.4.2 Routing Algorithm 2

Figure 2.5 described an algorithm to find a feasible control-layer routing solution, and to simultaneously minimize the channel length, latency, and skew. Because placement of control pins becomes progressively harder as more routing resources are occupied, the algorithm attempts to reserve routing space for each control pin after each successful routing step, even if more valves will be connected to the routing tree later. Although this method significantly increases the routability, the routing quality is decreased because the routing process is continuously disrupted by the area that is temporarily reserved for the remaining control pins. In this subsection, we proposed an enhanced methodology for the control-layer optimization problem. The proposed methodology handles the problems of control-pin placement and control-channel routing in a more global manner. The key idea is that a control pin is not assigned until the routing tree is completely established and optimized. Although the routing quality—quantified by the latency and channel length—is improved, the routability of this method decreases because fewer routing sources are available for control pin placement in the later stages of the optimization, which increases the likelihood of unroutable results.

In the second routing method (Algorithm 2), all valves for a given valve-addressing solution are again first partitioned into several groups. The valve-addressing solution is generated by mapping it to a minimum clique-partitioning problem (Sect. 2.2.3). Valves in the same group can be connected to one another and can share a single control pin without any control conflicts. Once valve addressing is completed, we establish the routing tree from a global perspective instead of routing the control channels line by line. Specifically, in the proposed methodology, all valves in the same group are separately pre-routed and optimized without considering their effects on other valves. Note that the latency of a routing solution depends only on the longest pressure-propagation path in the chip. By initially generating a pre-routed solution for each group, we can easily identify which groups are latency bottleneck, and we assign higher routing priorities to them in the subsequent routing attempts.

The group-based control-channel optimization problem described above can be mapped to the obstacle-avoiding rectilinear Steiner minimal tree (OARSMT) problem [19, 20]. Given a set of valves and a set of obstacles on a plane, an OARSMT connects these valves, possibly through some additional points (called Steiner points), and avoids passing through any obstacle to construct a tree with a minimal total channel length. Figure 2.9a illustrates the OARSMT problem. Green circles indicate valves. Black rectangles indicate obstacles. The control-channel routing problem is more complex than OARSMT because multiple Steiner trees (i.e., valve groups) need to be implemented simultaneously. Hence, during the routing of each Steiner tree, the interactions between the different trees must be considered. The latency of a chip is defined as the longest pressure-propagation in the chip, while the latency of a Steiner tree (i.e., a valve group) is defined as the longest path in this tree.

We present a graph-theoretic solution based on the method proposed in [21] to solve the above problem. Let $P = \{P_1, P_2, \dots, P_m\}$ be a set of valves for an m -

valve net, $O = \{O_1, O_2, \dots, O_k\}$ be a set of k obstacles, and $C = P \cup S$, where S is a set of corners of obstacles in O (Fig. 2.9a). Note that $|C| \leq m + 4k$ since each obstacle has four corners. The Manhattan distance between points with coordinates (x_i, y_i) and (x_j, y_j) is given by $|x_i - x_j| + |y_i - y_j|$.

The flow chart of the proposed Algorithm 2 is presented in Fig. 2.8. We next explain the flowchart step-by-step.

Steps 1–4: Same as for Algorithm 1 (Sect. 2.4.1).

Steps 5–7: Pre-route the chip to estimate the latency of each valve group. The groups with larger latencies will be given higher priorities in the final routing-tree implementation stage.

Step 5: Because the latency only depends on the longest pressure-propagation path, it is necessary to identify the critical valve group that contains the longest path, so that we can attempt to reduce the path length in the final routing attempts. In our algorithm, we generate an undirected connected spanning graph G (Fig. 2.9b), where the following two statements hold:

- all vertices in vertex set C are connected; and
- no edge intersects with an obstacle.

Liu [21] has demonstrated that such a spanning graph implies a rectilinear shortest obstacle-avoiding path between any two vertices in C . Hence, a path with the shortest Manhattan distance can be obtained by simply selecting some edges in G and transforming them into vertical and horizontal edges.

Step 6: A spanning tree T is established for each valve group. Figure 2.9c shows an example of the spanning tree. The weight of an edge is defined as the shortest Manhattan distance between its two vertices, which can easily be obtained using Dijkstra's shortest path algorithm [20]. Hence, each edge in the spanning tree can be mapped to a shortest path in the spanning graph.

Step 7: Prim's algorithm is used to establish the minimum spanning tree T' by selecting edges from the spanning tree T (Fig. 2.9d) [20]. After mapping it to the spanning graph, we can calculate the length of the longest pressure-propagation path (i.e., latency) in each tree after placing a control pin at the tree's center and use it to estimate the path length of each valve group. The latency of a chip depends only on the valve group (i.e., spanning tree) with the largest latency.

From Step 8, we begin to implement routing. We must globally consider the assignment of the routing resources to different valve groups. Furthermore, to reduce latency, the larger spanning tree should be assigned higher priority during routing. In our method, the valve groups are routed in descending order of their estimated latencies. Hence, more routing resources are available to valve groups with larger path lengths to ensure better routing quality. The routing procedure is similar to that presented in Step 6–7.

Step 8: The congestion and routing costs are initialized according to the pre-routing results obtained in Step 7. The methodology to calculate the routing cost is illustrated in Fig. 2.6. The congestion of an edge in the grid is the sum of the congestion generated from each valve group.

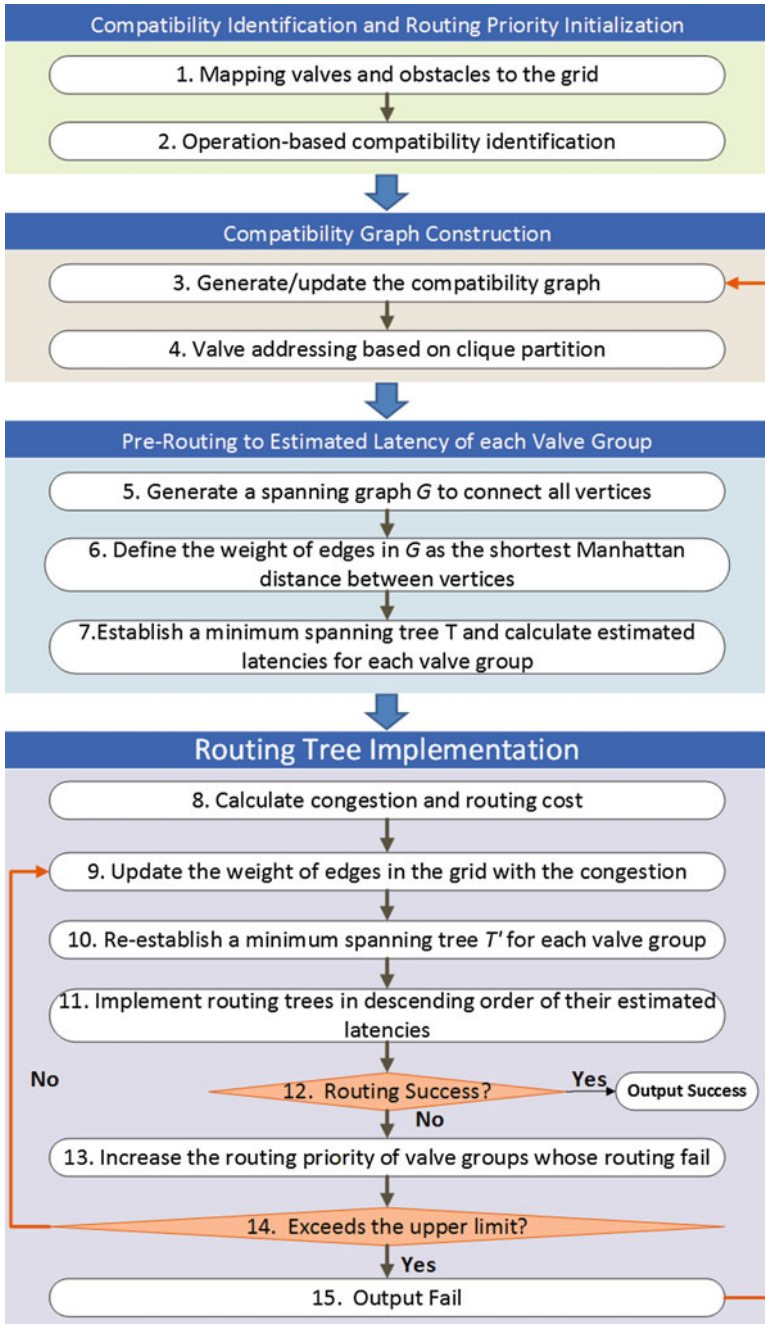
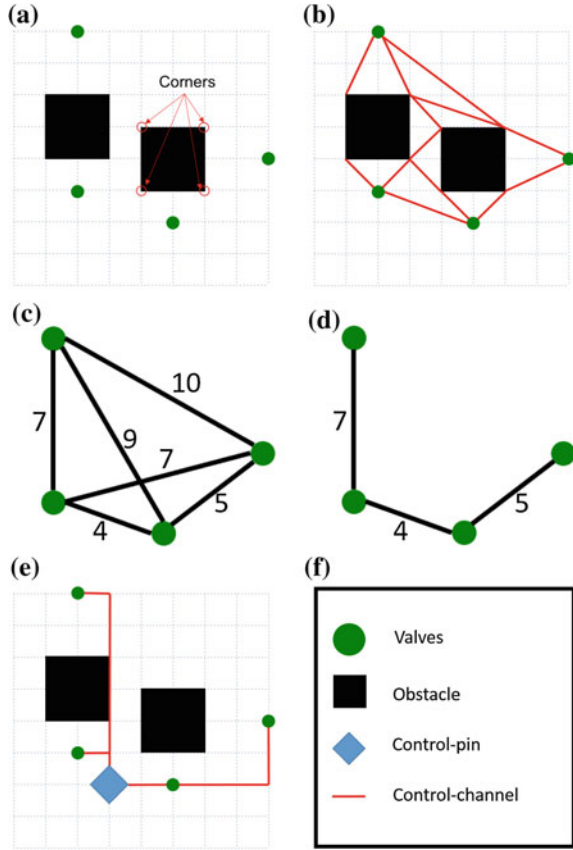


Fig. 2.8 Flowchart corresponding to Algorithm 2

Fig. 2.9 An example to illustrate Algorithm 2. **a** Illustration of the OARSMT problem. *Green circles* indicate valves. *Black rectangles* indicate obstacles. **b** The corresponding undirected connected spanning graph G generated in Step 5. **c** The corresponding spanning tree T . **d** The minimum spanning tree of T . **e** The corresponding spanning tree routing. **f** Legend



Step 9: We regenerate a spanning tree F , whose edge weight is defined as the minimal routing cost between its two end vertices. Dijkstra's shortest path algorithm can be applied to compute the minimal routing cost of a valve pair, since the routing cost is defined as the sum of the congestion value of all on-path grid edges.

Step 10: Similar to Step 7, Prim's algorithm is applied to obtain a minimum spanning tree F' . Because we can map each edge in F' to a path with the smallest routing cost in the spanning graph G , the total routing cost for all valve groups is minimized.

Step 11: In this step, we implement minimum spanning trees in descending order of their estimated latencies. The latency of each spanning tree is estimated in Step 7. The implementation is realized by transforming edges in F' to vertical/horizontal control channels in the grid. Moreover, [21] presents some local refinements to further improve the channel length. Note that after a valve group is routed, the congestion of each grid point on the chip should be updated by summing the congestion contribution of all unrouted valve groups, and, a control pin should be placed at the center of the

corresponding routing tree or at a nearby location. An example of spanning tree routing is shown in Fig. 2.9e.

Step 12–13: If all valve groups are successfully routed and their control pins are placed, the final routing result will be provided as output. Otherwise, the failed valve group will be given a higher priority. We return to Step 9 until all valve groups are routed.

Step 14–15: If there exists a valve group for which the number of routing attempts exceeds an upper limit, it will be labeled “unroutable”. To avoid any further attempts to route “unroutable” valve groups, the compatibility graph will be updated by removing the edge with largest routing cost in the corresponding F' . As a result, valve addressing based on the new compatibility graph will divide this valve group into two groups.

Finally, we consider an extreme case of the control-channel routing problem. If a chip consists of a valve group with a large path length and several valve groups with much smaller path lengths, the latency of this chip will depend only on the largest valve group. As a result, we should minimize the path length of the large valve group rather than the routing cost during the routing implementation. Hence, in our algorithm, the congestion of each edge c' in the grid is actually defined as $\sqrt[n]{c}$, where c is the regular congestion value presented in Fig. 2.6, and n is the variation of the path length of all valve groups. In the extreme case, $n \Rightarrow \infty$, therefore $c' \Rightarrow 1$, which is the Manhattan distance of each edge in the grid. Thus, the edge weight of the spanning tree F is transformed from the sum of congestion between vertices (i.e., routing cost) to the minimal Manhattan distance between vertices.

2.5 Experimental Results

We perform two sets of simulations to demonstrate the effectiveness of the proposed optimization method. The evaluation is carried out on the basis of four metrics: the number of control pins, total channel length, latency, and skew. *Latency* is the delay caused by the longest pressure-propagation path, i.e., the minimum allowable pattern set-up time. Given the longest routing path in the design, P , the latency can be defined as: $T_{Latency} = P/v$, where v is the velocity of pressure propagation. In this simulation, v is set to 10 mm/s in all cases [10]. *Skew of a group* is the maximum difference among the pressure-signal arrival times of all valves in this group. Given a design with n control pins (and hence n valve groups), the skew of a group i , referred to as $T_{skew,i}$ can be defined as: $T_{skew,i} = (P_{l,i} - P_{s,i})/v$, where $P_{l,i}$ and $P_{s,i}$ are the longest and shortest pressure-propagation paths in this group, respectively. The *total skew* can be defined as the maximum skew over all the groups, i.e., $\max_i \{T_{skew,i}\}$. Furthermore, we provide the pin count required by the table-based compatibility identification method [13] in order to evaluate the efficiency of the proposed operation-based approach. Although the operation-based method requires a few more control pins, it ensures that the designed chips remain fully functional even if users decide to customize their experiments during the on-chip execution of

biochemistry procedures. It is difficult to quantify the increased flexibility provided by the proposed method over [13], but the discussion in Sect. 2.2.3.1 has highlighted this issue in detail.

In the first set of experiments, we evaluate the proposed optimization method using two fabricated, but manually-designed, flow-based mVLSI biochips [22, 23]. The results of our approaches are compared with the actual implemented design of these real-life chips. In the second experiment, we generate six synthetic benchmarks and compare the results derived by the proposed methods with a baseline method. Based on the experimental results, we conclude that by taking the valve addressing, control-channel routing and control-pin placement into consideration in a unified manner, the proposed method leads to superior designs compared to the results obtained earlier by manual optimization, as well as the baseline automated method. Furthermore, Algorithm 2 leads to a better routing quality in most cases, but Algorithm 1 has a higher likelihood of finding a routable solution. The proposed methods were implemented in C++ and experiments were carried out on an Intel Core i5 2.67 GHz Linux machine with 16 GB Memory. The CPU time for each test case was less than 50 s.

2.5.1 Experiments with Two Fabricated Biochips

We considered two fabricated chips, whose layouts are shown in Fig. 2.10, were targeted for chromatin immunoprecipitation (ChIP). Chip I allows more control measurements and therefore has higher sensitivity [22], while Chip II allows more reaction volume and therefore provides higher throughput [23]. Both chips are constructed by placing and interconnecting multiple copies of two basic components, i.e., multiplexer and mixer, whose component actions are listed in Fig. 2.3c, d. By combining the component actions, we can obtain a complete set of functional actions required by these two chips.

The comparison of design quality between these fabricated designs and the proposed automated methods are reported in Table 2.1, where “TBCI” denotes the the number of control pins required by the baseline table-based compatibility identification method. Although the proposed OBCI method requires three more control pins for each case (for both Algorithm 1 and Algorithm 2), the designed chips are fully functional and are able to provide much-needed experimental flexibility for realistic scenarios. Furthermore, for Chip I, there is no feasible routing solution for the design generated using TBCI.

Compared with manual optimization, although the skew and latency of our method are slightly worse, the routing results of proposed methods show significant improvements in terms of the pin count and total channel length in both test cases. The manual design of Chip I (as actually fabricated) led to 72 control pins for the 176 valves, while the proposed automated methods needs only 40 pins (44.4% reduction). Moreover, to shorten the response time for valve actuation, the proposed Algorithm 1 and Algorithm 2 reduce the channel length by about 60%. A significant benefit in terms

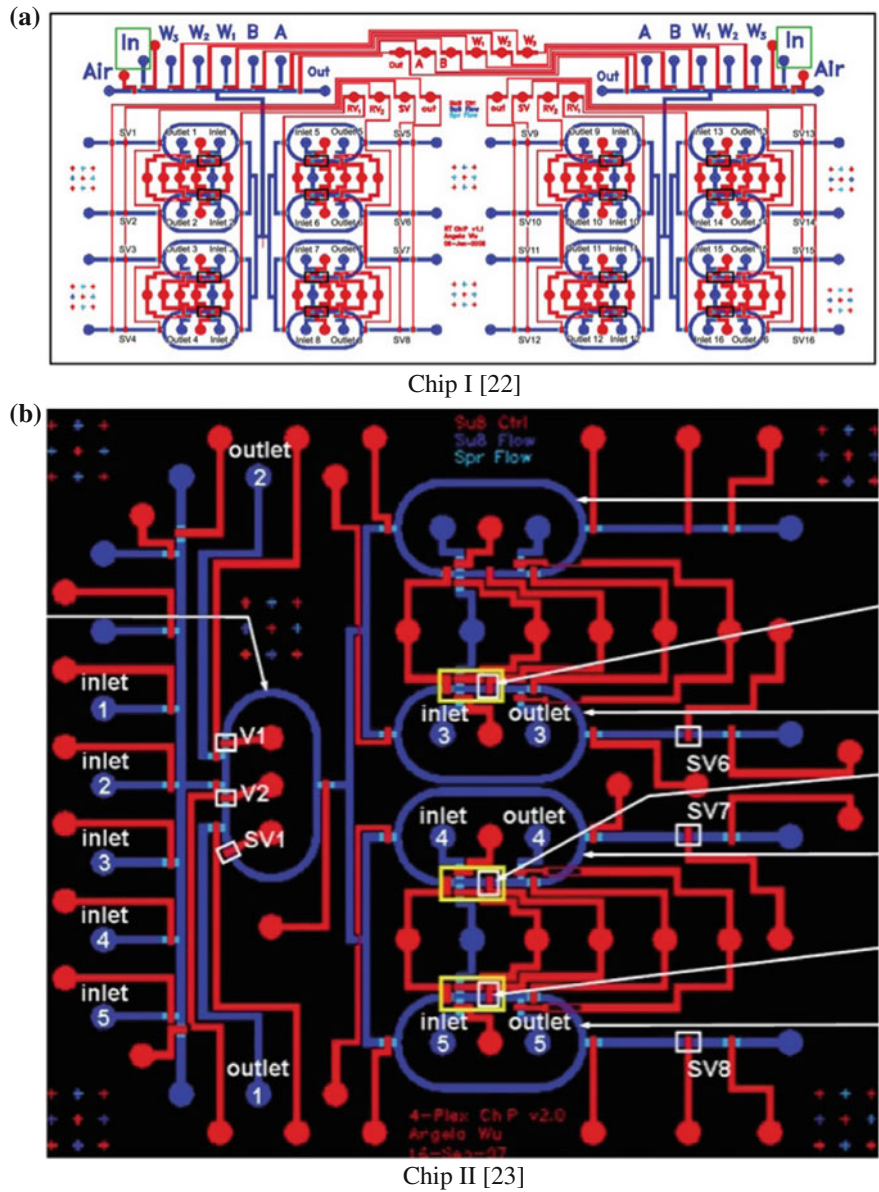


Fig. 2.10 Layouts of the fabricated chips used for evaluation. **a** Chip I; **b** Chip II

Table 2.1 Comparison of design quality with manual design for two fabricated chips. “W” denotes width of biochips, “L” denotes length of chips, “Length” denotes total channel length, “TBCI” denotes the number of control pins required by the table-based compatibility identification method

Chip #	Specification			Manual optimization					OBCI (Algorithm 1)					OBCI (Algorithm 2)					TBCI				
	W (mm)	L (mm)	# Valves	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)
Chip I	36	81	176	72	2613	0.33	5.767	40	1066	0.749	6.331	40	978	1.328	5.631	37	NR*	NR*	NR*	37	NR*	NR*	NR*
Chip II	36	41	56	46	398.2	0	1.847	34	212.2	0.085	1.900	34	188.5	0.105	1.830	31	156.2	0.073	1.082				

NR*: Design not routable

Table 2.2 Comparison of design quality with a baseline method based on six synthetic benchmarks. The number of control pins for TBCI is slightly less but there is significant loss in on-chip biochemistry functionality

Bench mark #	Specification			Baseline method					OBCI (Algorithm 1)					OBCI (Algorithm 2)					TBCI				
	W (mm)	L (mm)	# Valves	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)	# Pins	Length (mm)	Skew (s)	Latency (s)
1	5	5	4	3	18	0.29	0.55	3	15.3	0	0.48	3	15.3	0	0.48	3	15.3	0	0.48	3	15.3	0	0.48
2	10	10	10	3	NR*	NR*	NR*	3	53	0.58	1.5	3	49	0.58	1.5	3	53	0.58	1.5	3	53	0.58	1.5
3	25	25	15	8	125	1.57	3.55	5	185	0.47	3.9	5	170	0.47	3.1	5	185	0.47	3.9	5	185	0.47	3.9
4	35	35	20	9	262	2.00	8.15	9	224	0.65	3.9	9	NR*	NR*	NR*	8	NR*	NR*	NR*	8	NR*	NR*	NR*
5	75	75	40	14	NR*	NR*	NR*	14	1435	2.69	12.9	14	NR*	NR*	NR*	11	NR*	NR*	NR*	11	NR*	NR*	NR*
6	72	81	288	84	NR*	NR*	NR*	92	1624	0.085	3.4	92	1497	0.105	3.4	84	NR*	NR*	NR*	84	NR*	NR*	NR*

NR*: Design not routable

of these two metrics can be observed for Chip II as well. The experimental results also show that the proposed OBCI method enables the designed biochips to be fully functional and flexible at the cost of only three additional control pins in each case. Furthermore, the routing quality for Algorithm 2 is slightly better than that of Algorithm 1 in both cases.

2.5.2 Experiments with Synthetic Benchmarks

To further demonstrate the effectiveness and applicability of the proposed method, we simulate five applications and generate five corresponding synthetic chip layouts (Benchmark 1–5). In addition, we have created an artificial larger test case by combining Chip I and Chip II discussed in Sect. 2.5.1 (Benchmark 6). Note that the length of Chip I is twice of Chip II. Hence, in this hypothetical design, two copies of Chip II, whose layout is shown in Fig. 2.10b are first combined, and then one copy of Chip I is combined with this layout. In addition, a baseline method is implemented and compared with the proposed approach. In the baseline method, we first implement a heuristic algorithm for valve addressing [13]. After that, we use a maze-routing algorithm based on breadth-first search to sequentially route the valves that share the same control pin, and then check all grid points along the routing path to find a possible placement of the control pin. This procedure is repeated until all valves are connected to the control pins or an unroutable decision is made.

The design quality of the baseline method, and the proposed Algorithms 1 and 2 are compared in Table 2.2. The baseline method cannot find feasible routing solutions for Benchmarks 2, 5, and 6; however, Algorithm 1 successfully addresses all test cases. Moreover, Algorithm 1 also achieves a 15% reduction in the number of pumps, a 36% reduction in total channel length, a 76% reduction in skew, and a 33% reduction in latency on average for the completed test cases (Benchmarks 1, 3, and 4). In addition, Algorithm 2 leads to reduced channel length and latency for Benchmarks 2, 3, and 6, although it fails to find routing solutions for Benchmarks 4 and 5. Furthermore, the number of control pins required by the two compatibility identification methods are compared. The operation-based approach can achieve maximum flexibility with only a few additional control pins. More importantly, the low pin-count provided by TBCI leads to infeasible designs, since these designs are not routable.

2.6 Conclusions

We have presented the first practical problem formulation for automated control-layer design in flow-based microfluidic VLSI (mVLSI) biochips and described a systematic optimization approach for solving this problem. The goal of this optimization technique is to find an efficient routing solution for control-layer design with a minimum number of control pins. We have described an operation-based compatibility

identification method that obviates the need to exhaustively list activation sequences. By independently targeting the fluid-operation actions of each on-chip component, the proposed methods not only decreases the search space, but it also ensures that all possible fluid-handling operations can be executed. The pressure-propagation delay, an intrinsic physical phenomenon in mVLSI biochips, has been minimized in order to reduce the response time for valves, decrease the pattern set-up time, and synchronize valve actuation. Two fabricated flow-based devices and six synthetic benchmarks have been used to evaluate the proposed optimization method. We have shown that, compared with manual control-layer design and a baseline approach, the proposed approach leads to fewer control pins, better timing behavior, and reduced channel length in the control layer.

References

1. K. Hu, T.A. Dinh, T.-Y. Ho, K. Chakrabarty, Control-layer optimization for flow-based mvlsi microfluidic biochips, in *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, ser. CASES '14 (ACM, New York, NY, USA, 2014), pp. 16:1–16:10. <http://doi.acm.org/10.1145/2656106.2656118>
2. K. Hu, T.A. Dinh, T.-Y. Ho, K. Chakrabarty, Control-layer routing and control-pin minimization for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **36**(1), 55–68 (2017)
3. W.H. Minhass, P. Pop, J. Madsen, System-level modeling and synthesis of flow-based microfluidic biochips, in *Proceedings International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. IEEE, 2011, pp. 225–233
4. W.H. Minhass, P. Pop, J. Madsen, F.S. Blaga, Architectural synthesis of flow-based microfluidic large-scale integration biochips, in *Proceedings International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*. ACM, 2012, pp. 181–190
5. N. Amin, W. Thies, S. Amarasinghe, Computer-aided design for microfluidic chips based on multilayer soft lithography, in *IEEE International Conference on Computer Design*. IEEE, 2009, pp. 2–9
6. J. Melin, S.R. Quake, Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annu. Rev. Biophys. Biomol. Struct.* **36**, 213–231 (2007)
7. P. Tabeling, *Introduction to Microfluidics* (Oxford University Press, 2010)
8. D.C. Leslie, C.J. Easley, E. Seker, J.M. Karlinsey, M. Utz, M.R. Begley, J.P. Landers, Frequency-specific flow control in microfluidic circuits with passive elastomeric features. *Nat. Phys.* **5**(3), 231–235 (2009)
9. M. Tanyeri, M. Ranka, N. Sittipolkul, C.M. Schroeder, A microfluidic-based hydrodynamic trap: design and implementation. *Lab Chip* **11**(10), 1786–1794 (2011)
10. Y. Lim, A. Kouzani, W. Duan, Lab-on-a-chip: a component view. *Microsyst. Technol.* **16**(12), 1995–2015 (2010)
11. T. Thorsen, S.J. Maerkl, S.R. Quake, Microfluidic large-scale integration. *Science* **298**(5593), 580–584 (2002)
12. Stanford Microfluidic Foundry: Basic Design Rules, <http://www.stanford.edu/group/foundry/Basicaccessed:%20Design%20Rules.html>. Accessed 05 Jan 2017
13. W.H. Minhass, P. Pop, J. Madsen, T.-Y. Ho, Control synthesis for the flow-based microfluidic large-scale integration biochips, in *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*. IEEE, 2013, pp. 205–212
14. R. Gómez-Sjöberg, A.A. Leyrat, D.M. Pirone, C.S. Chen, S.R. Quake, Versatile, fully automated, microfluidic cell culture system. *Anal. Chem.* **79**(22), 8557–8563 (2007)

15. T. Xu, K. Chakrabarty, Broadcast electrode-addressing for pin-constrained multi-functional digital microfluidic biochips, in *Proceedings ACM/IEEE Design Automation Conference*. ACM, 2008, pp. 173–178
16. J.L. Gross, J. Yellen, *Graph Theory and its Applications* (CRC Press, 2005)
17. C.Y. Lee, An algorithm for path connections and its applications. *IEEE Trans. Electron. Comput.* **3**, 346–365 (1961)
18. D. Brélaz, New methods to color the vertices of a graph. *Commun. ACM* **22**(4), 251–256 (1979)
19. Y. Hu, Z. Feng, T. Jing, X. Hong, Y. Yang, G. Yu, X. Hu, G. Yan, Forst: A 3-step heuristic for obstacle-avoiding rectilinear steiner minimal tree construction. *J. Inf. Comput. Sci.* **1**(3), 107–116 (2004)
20. T.H. Cormen, *Introduction to Algorithms* (MIT Press, 2009)
21. C.-W. Lin, S.-Y. Chen, C.-F. Li, Y.-W. Chang, C.-L. Yang, Obstacle-avoiding rectilinear steiner tree construction based on spanning graphs. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **27**(4), 643–653 (2008)
22. A.R. Wu, T.L. Kawahara, N.A. Rapicavoli, J. van Riggelen, E.H. Shroff, L. Xu, D.W. Felsher, H.Y. Chang, S.R. Quake, High throughput automated chromatin immunoprecipitation as a platform for drug screening and antibody validation. *Lab Chip* **12**(12), 2190–2198 (2012)
23. A.R. Wu, J.B. Hiatt, R. Lu, J.L. Attema, N.A. Lobo, I.L. Weissman, M.F. Clarke, S.R. Quake, Automated microfluidic chromatin immunoprecipitation from 2,000 cells. *Lab Chip* **9**(10), 1365–1370 (2009)

Computer-Aided Design of Microfluidic Very Large Scale
Integration (mVLSI) Biochips

Design Automation, Testing, and Design-for-Testability

Hu, K.; Chakrabarty, K.; Ho, T.-Y.

2017, XIII, 142 p. 64 illus., 55 illus. in color., Hardcover

ISBN: 978-3-319-56254-4