A Pin-Count Reduction Algorithm for Flow-Based Microfluidic Biochips

Alexander Schneider, Paul Pop, Jan Madsen
Tecnical University of Denmark
{alsch, paupo, jama}@dtu.dk

***Abstract*—Microfluidic biochips are replacing the conventional biochemical analyzers integrating the necessary functions on-chip. We are interested in flow-based biochips, where a continuous flow of liquid is manipulated using integrated microvalves, controlled from external pressure sources via off-chip control pins. Recent research has addressed the physical design of such biochips. However, such research has so far ignored the pin-count, which rises with the increase in the number of microvalves. Given a biochip architecture and a biochemical application, we propose an algorithm for reducing the number of control pins required to run the application. The proposed algorithm has been evaluated using several benchmarks.**

I. Introduction

Microfluidics refers to a technology that miniaturizes biological and chemical processes to a sub millimeter scale. A biochip integrates different biochemical functionalities such as mixers, filters and detectors on a single chip, leading to higher portability, throughput and sensitivity, while reducing sample volume consumption [2]. Microfluidic large-scale integration (mLSI) enables the development of microfluidic chips using hundreds of such functions, allowing multiple assays to be run in parallel, making them usable for tasks such as Protein Crystallography, Amino Acid Analysis or Chemical Synthesis [5]. The key for complex functionality on biochips is the use of on-chip valves, similar to transistors in semiconductor LSI. Such valves are manufactured using multilayer soft lithography and are controlled by outside pressure sources [7]. Using these valves the flow of fluid within the chip can be restricted, allowing to decide if and in which order the functions on the chips are used.

Ongoing research enables the fabrication of increasingly complex biochips, with the number of valves integrated on a single chip reaching the thousands [4]. Through these advancements, current methodologies for chip design, flow, and control synthesis become inadequate [9]. In this paper we focus on part of the control synthesis problem, namely the control pin reduction problem. A control pin is a physical hole in the chip, which provides access to the control layer. By applying pressure (or vacuum, depending on the valve type) [11] to the control pin, a valve on the chip can be activated. However, a large number of control pins is infeasible, due to the resulting consumption of chip area and required bulky off-chip control. The 'AssayMark' controller from Microfluidic Innovations for example, is only capable of providing pressure to up to 36 individual control pins, using solenoid valves [3]. By allowing valve sharing, i.e. the sharing of the same control by multiple valves, it is thus possible to reduce the required amount of control pins to meet these restrictions. Doing so reduces the flexibility of the chip, since the shared valves will work in unison. Valves sharing a single control pin therefore have to be chosen carefully, in order to keep the chip operable [7].

Researchers have previously proposed approaches to the application mapping and scheduling [8]. Based on this schedule of operations, the control information (which valves to open and close at what time and for how long) can be extracted. Using this information, optimization schemes can be applied to minimize the chip's pin-count in the control layer. Recent research has proposed approaches to control pin minimization following this idea [2] [7] [10]. All this related work however uses simplified routing, which directly affects the pin-count reduction. Furthermore, the ordering in which scheduling and pin-count reduction is executed is crucial, since the execution of either process poses constraints onto the other. Combining valve controls and therefore having valves work in unison imposes scheduling constraints on such routes, which require these valves to be in opposing states. Vice versa, if the schedule has already been completed and such routes are to be executed at the same time, valve control combinations are restricted to valves which are never in opposing states. Reducing the pin-count beforehand would therefore allow to trade off flexibility during scheduling for additional combinations.
Contribution: We propose a new pin-count reduction technique which is applied before the application is scheduled. Using information from the applications routing and binding instead of its schedule (which also incorporates timing constraints) allows for additional combinations for controls to be found. If required, this enables combinations to be made which potentially increase the schedule length, resulting in the possibility for a direct trade-off between the pin-count and the applications execution time. To improve the pin-count reduction, we also propose a new routing technique which prioritizes routes which maximize the potential for control combinations.

II. Biochip Architecture Model

Fig. 1 shows a functional view of a flow-based biochip containing multiple **Functional Fluid Units (FFU)** such as inputs and heaters, as well as switches connecting the channels leading to these components. The basic building block of these

978-1-5090-1457-6/16/$31.00 ©2016 IEEE

components is a microvalve, which can be used to manipulate the fluid flow.

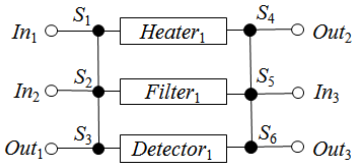


Fig. 1. Architecture

Fig. 2 shows a micro-mechanical valve which can be manipulated by an external force to either restrict or permit the fluid flow. To do so, the chip is logically divided into two layers, the flow-layer (colored in blue) which contains the fluid and the control-layer (colored in red) which can manipulate the flow-layer [10]. To create functional components, multiple valves are needed. Fig. 3a shows two variations of switches, requiring three and four valves respectively to control the path of the fluid entering from any side. Mixers, as shown in Fig. 3b, require nine valves to be operational. Other components such as filters, heaters or detectors only require two valves to close off the component during its execution, similar to valve 1 and 8 in the mixer [1].

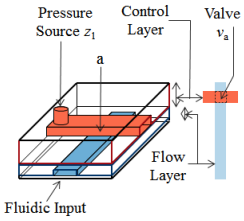
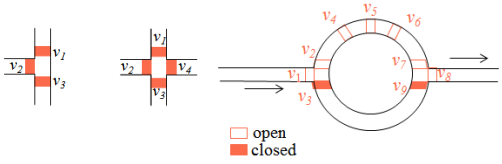


Fig. 2. Micro-mechanical Valve.



(a) Switches (b) Mixer

Fig. 3. Components

Each valve is in one of two physical states. The *open* state allows for fluid to be transport through a channel or component and the *closed* state prevents fluid from leaking into other channels or components. Additionally, we also distinguish a third, logical state we call *don't care*. While valves in the *open* or *closed* state affect the fluid transport by allowing or restricting the flow respectively, valves in the *don't care* state have no effect on the fluid transport at all, making their physical state irrelevant. Consider the mixer in Fig. 3b: To fill the bottom half, valves 1, 3, 8 and 9 have to be open to allow fluid transport, while valves 2 and 7 have to be closed to prevent leakage into the top half. The state of valves 4, 5, and 6 however is irrelevant, since the channel they are located in can not be reached by any fluid when valves 2 and 7 are closed, placing them in the *don't care* state.

A. Architecture Model

We make use of the system-layer architecture model based on a topology graph, as proposed in [8]. Fig. 1 shows an example of a biochip architecture, containing three inputs and outputs, one filter, one heater and one detector. We distinguish between two kinds of vertices in this model: Switches, which create intersections between multiple channels (e.g. S_1) and FFUs which can perform a certain action (e.g. In_1, $Heater_1$, $Filter_1$). Edges represent channels through which fluid can be transported and are considered bi-directional in this paper. Fluid can be transported through these components by applying pressure to an input from an outside source. A functional route must therefore always start at an input where pressure is applied and end in an output where the pressure is released from the chip. To identify such routes, we introduce the following notations.

A **Flow Path Segment (FPS)** is a set of vertices, forming a directed route from one FFU to another. To connect the two FFUs, a FPS can contain any number of switches, but no additional FFUs, e.g. (In_1, S_1, S_2, $Filter_1$). FPSs are mutually exclusive, meaning they cannot be used to transport fluids at the same time, if they share at least one vertex.

A **Flow Path (FP)** is a set of FPSs which form a route usable for fluid transport within the given architecture. The first FPS of such a set has to start with an input, since they act as a pressure source. All following FPSs have to start where the previous FPS ended. A FPS ending in an output completes the FP which then forms a continuous route from an input to an output. Considering the architecture in Fig. 1, a valid FP would for example be: ((In_1, S_1, $Heater_1$), ($Heater_1$, S_4, Out_2))

Previous research [2] [8] has simplified routing, by assuming

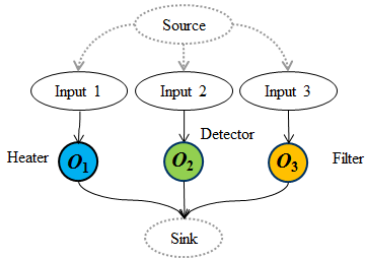


Fig. 4. Application

that fluids can be moved along all given FPSs, regardless of the presence of inputs or outputs. This only works under the assumption of implicit inputs and outputs, which would be located in front of and behind FFUs, allowing FPSs to be functional routes on their own. We refer to these additional interfaces as implicit, since they are not modelled in the architecture as opposed to explicit ports, which are part of the model. This results in invalid schedules, unless all additionally

assumed inputs and outputs are added to the architecture, which is likely to be infeasible due to the increase in valve- and control count as well as the chip size.

B. Application Model

To model a biochemical application, we use a directed, acyclic and polar sequencing graph, as explained in detail in [8]. For simplicity reasons, we omit operation specific execution times and assume that it takes 1 time step for fluid to pass through any given channel (e.g. from one switch to another) and 4 time steps for an operation (e.g. mixing) to finish. An example of such an application graph can be seen in Fig. 4.

III. Problem Formulation

Let us consider the In-Vitro Diagnostics application (IVD) from Fig. 6 to be mapped on the architecture from Fig. 5. One possible, partial schedule for IVD is shown in Fig. 8 and Fig. 7 shows the valve actuation sequence according to this schedule. From this table, possible control combinations can be found as suggested in the related work [7]: Valve Groups 1 and 3 for example, can be in the same state throughout the depicted time steps (at time step (TS) 18, VG 3 is in a don't care state and can be opened to suit VG 1) and their controls can therefore be combined.

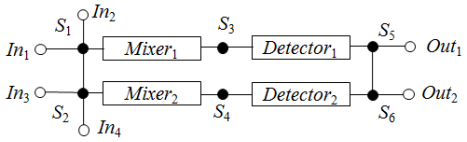


Fig. 5. IVD Architecture

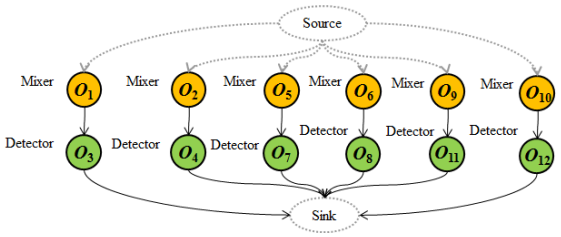


Fig. 6. Typical IVD application that mixes various samples, reagents and buffers and analyses the results.

VG No. / TS	8	10	12	14	16	18	20	22
1	0	0	0	0	0	0	X	X
2	1	1	0	0	1	1	X	X
3	0	0	0	0	0	X	X	X
4	1	0	0	1	1	X	X	X
5	0	0	1	1	0	0	X	X
6	0	1	1	0	0	X	X	X
7	0	0	X	X	0	0	1	1
8	0	X	X	0	0	1	1	X
...	...	...	...	...	...	...	...	...
8A	0	X	X	0	0	X	1	1

Fig. 7. Partial actuation Table for IVD. 0: Open. 1: Closed. X: Don't Care

In IVD, the detector operations O3 and O4 are waiting for the mixing operations O1 and O2 to finish respectively. The fluid transport M2 and the following detector operation O4 are started as soon as possible as it is expected to minimize the schedule length. This however means, that the equivalent operations for Detector_1 (O1, M1) are not executed at the same time as for Detector_2. This leads to the valves used by these four operations to be in opposing states in time step 18 as highlighted with the corresponding color coding in Fig. 7, meaning their controls can't be combined.
However, when combining the controls, e.g. for valves used by Detectors 1 and 2, before scheduling the operations, we are still able to schedule the application with only minor changes, as shown in Fig. 9. This control combination imposes the constraint that the detectors can't be used asynchronously, which is addressed by postponing the execution of operation O4 until operation O3 can be executed as well. The resulting change in the control logic for VG 8 is stated as VG 8A in Fig. 7, which is now compatible with VG 7. Postponing this operation does however also mean that all operations depending on the result of O4, are also potentially executed at a later time as well, which can increase the overall schedule length.
The problem can therefore be defined as follows:
Input: Application, architecture and valid routes for the provided architecture and application; state of all valves for each route, such that the route can be used and the available number of controls provided by the external hardware.
Determine: The sharing of valves, the schedule and routing of operations, such that the application does not use more than the provided number of control pins and the schedule length is minimized.

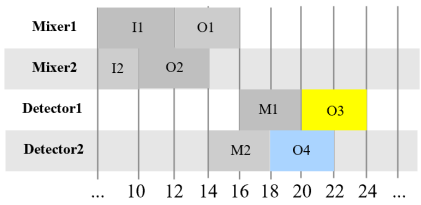


Fig. 8. Partial IVD schedule, created before pin-count reduction.

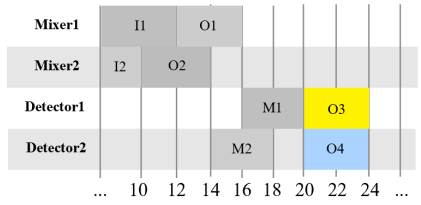


Fig. 9. Partial IVD schedule, created after pin-count reduction with scheduling constraints in place.

IV. Proposed Method

We introduce a novel routing technique, which creates a route for every operation given by the application, in ac-

cordance to the architecture model. The determined routes are then used as input for the proposed pin-count reduction algorithm along with the available number of controls. With the control combinations in place the application is scheduled, taking any scheduling constraints introduced during the pin-count reduction into account. We do not discuss scheduling in this paper, as it is solely used to compare the results. We have implemented a List-Scheduling algorithm [6] to schedule all experiments.

A. Routing

Given the architecture and application models (see Fig. 1 and Fig. 4 respectively), the routing algorithm creates a route for every operation. Every route consists of one or more FPs, which determine the path on which the fluid is transported through the chip. While all available FPSs can be determined from the architecture model, the information about the required FPs is stated by the application. Each operation of an application tells us where the fluid is currently located (e.g. Input 1) and to what kind of component (e.g. Heater) it needs to be routed to. We therefore need to find a path between these FFUs using a set of FPSs. We begin with all FPSs that start at an input, for example In_1, and check if any of them end at a heater. If so, this part of the route is completed. If no such FPS is available, we continue the search in a Breadth-First-Search fashion, linking multiple FPSs together until the target is reached. Once a connection has been found, we need to complete the FP by making sure the route starts at an input and ends in an output. Since we route towards the heater, we search for a path connecting the heater and any output, making sure that no conflicts with the path from In_1 to the heater are introduced. As this route already started from an input, the FP is completed. Depending on the architecture, it is possible that multiple paths connecting the source and target FFU exist or even multiple components of the same type (e.g. Input, Heater) are available. In either case, we choose the route, which blocks the least number of FFUs and switches on the chip. A component is considered blocked if it is actively used (i.e. routed through), or if it cannot be used by another route, since an adjacent component is used. This also means, that a blocked component's valves are either in the *open* or *closed* state. A route with fewer blocked components allows for a higher flexibility while scheduling since more components are still available to be used by another route.

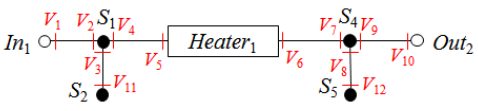


Fig. 10. Valves

B. Reducing the Pin-Count

Using the previously defined routes as input we can determine controls that can be combined to lower the chip's pin-count. As mentioned previously, combining controls without having a schedule in place provides additional possibilities for combinations, but also has the potential to increase the schedule length. This is due to scheduling constraints that are introduced when combining controls. The reason for the increase in schedule length is the same for all constraints: FPs that were previously able to be executed in parallel, now have to be executed in sequence to avoid unintentional mixing. The increase in schedule length however can vary, depending on the length of the FPs and the number of FPs that have to be executed in sequence because of the introduced constraint. Cases of such constraints will be shown using examples from Table I. Consider Fig. 1 as an example architecture. To operate the chip 32 valves are required. A partial example of where those valves are located in our architecture is shown in Fig. 10. The first set of combinations performed by the algorithm differ from other combinations, as they do not introduce any constraints onto the schedule. Those combinations can be found, regardless of the architecture, whenever multiple valves are located in the same channel. Consider In_1 and S_1 in Fig. 10. They are connected by a channel which contains valves 1 and 2. There exist only two ways in which this channel can be used: Either the channel is routed through, in which case both valves have to be opened, or it is not routed through, in which case both valves can be closed. The same principle applies to the valves which close off any given FFU (e.g. valves 5 and 6 in Fig. 10). Therefore, the valves' controls can be combined and share a control-pin, in what we call a **Valve Group (VG)**. Fig. 11 shows the result of applying this to the presented architecture, where the previously mentioned valves 1 and 2 now form VG_1, valves 4, 5, 6 and 7 form VG_2 and so on. Further combination of VGs means that all valves of both groups share a single control-pin and are therefore activated in unison.

Other types of combinations will introduce scheduling constraints, which may negatively affect the schedule length. To reach the objective of minimizing the schedule length, it is

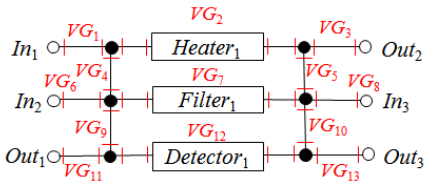


Fig. 11. Valve Groups

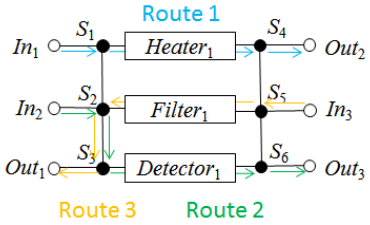


Fig. 12. Routes

TABLE I
VALVE GROUP COMBINATION EXAMPLES

Example	Combined VGs	Result	Reason
1	VG_1 and VG_4	Route 1 invalidated	VG_1 has to be open to allow fluid transport. VG_4 has to be closed to prevent leakage
2	VG_1 and VG_2	No increase in schedule length	VG_1 and VG_2 are *open* for Route 1 and in a *don't care* state for Routes 2 and 3. Therefore no matter which routes are executed in parallel, VG_1 and VG_2 can always be open
3	VG_6 and VG_{13}	No increase in schedule length	Even though this combination does prohibit routes 2 and 3 from being executed in parallel, no increase in schedule length is possible since these routes were not able to run simultaneously in the first place, due to their partially overlapping FPs
4	VG_1 and VG_{12}	Possible increase in schedule length	VG_1 has to be open for Route 1, while VG_{12} has to be closed for Route 3, prohibiting parallel execution of these routes
5	VG_1 and VG_{10}	Possible increase in schedule length	VG_1 has to be open for Route 1, while VG_{10} has to be closed for routes 2 and 3, prohibiting parallel execution of these routes

Example	Constraint	Result	Schedule length
6	Routes 1 and 2 restricted from being executed at the same time	Routes 1 and 3 are executed in parallel Route 2 is executed as soon as Route 1 finishes	18
7	Routes 1 and 3 restricted from being executed at the same time	Routes 1 and 2 are executed in parallel Route 3 is executed as soon as Route 1 finishes	16

TABLE II
VALVE GROUP CONFIGURATION

Route	Open VGs	Closed VGs	Don't Care VGs
1	1, 2, 3	4, 5	6, 7, 8, 9, 10, 11, 12, 13
2	6, 9, 12, 13	4, 7, 10, 11	1, 2, 3, 5, 8
3	7, 8, 9, 11	4, 5, 6, 10, 12	1, 2, 3, 13

therefore necessary to determine combinations that introduce constraints with minimal effect on the schedule. To clarify the effects of such constraints we consider the application from Fig. 4 on the architecture in Fig. 1. The three resulting routes can be seen in Fig. 12.

From these routes we can extract the valve states for each route, which is illustrated in Table II. Each route has sets of VGs in the previously explained states *open*, *closed* and *don't care*. Using this data we can determine the potential effect on the schedule length for each combination of VGs. Since a combination of any VGs is generally possible, an algorithm determines the potential effect of every possible combination of VGs. This algorithm first checks two special cases, one occurs if the combination invalidates any of the given routes. This is the case whenever one VG is in the *open* state, while the other is in the *closed* state for any of the routes, as shown in Example 1 in Table I. The second special case occurs when all valves can be in the same state for all routes, i.e. both VGs are either in the *open* or *don't care* state, or in the *closed* or *don't care* state, for all routes. If so, no conflict during parallel execution can arise and the schedule length can't be affected. Example 2 shows such a case. For the rest of the combinations the algorithm checks, for each pair of routes, whether the scheduling constraint introduced by the current combination would prohibit previously possible parallelism. Hence, we determine if the two routes were parallelizable in the first place, as some routes can't be executed at the same time, since they partially overlap as demonstrated in Example 3. Furthermore a constraint can only prohibit the parallel execution of two routes, if they require the combined VGs to be in opposing states. Example 4 states such a case, where routes 1 and 3 can not be executed in parallel if this combination is made. When a case such as in Example 4 arises, we determine how much this combination and the resulting scheduling constraint affects the schedule length. Exact results can however only be gathered by scheduling the application and comparing the schedule lengths. Since this is too time consuming, we propose a cost function to predict the affect on the schedule length. This prediction is based on how many parallel executions of routes are prohibited by a scheduling constraint. Example 5 prohibits route 1 from being executed at the same time as either route 2 or 3. Example 4 on the other hand only prohibits parallel execution of routes 1 and 3, leaving the possibility to execute routes 1 and 2 at the same time, leading to a prediction of a smaller increase in schedule length than for Example 5. Additionally the execution time of each route (how long it takes to transport the fluid along the FP) is taken into account. The function predicts that constraints affecting routes with long execution times have a larger effect on the schedule length. This can be seen in examples 6 and 7 where we assume that routes 1-3 have an execution time of 10, 8 and 6 respectively. Both examples prohibit one pair of routes from parallel execution, yet Example 7 results in a shorter schedule length, since the longest routes are not affected by constraints. Using this prediction we can directly compare the potential impact of any combination on the schedule length, allowing for a specific order in which the combinations are applied, until the desired number of required control pins is reached. As explained previously, the algorithm determines and discards combinations that would invalidate one or more routes. It is however possible to find alternate routes for such cases, which would allow these additional combinations to be made. This is possible if an alternate route can be found, which is valid with all previously determined combinations left in place. Finding such routes however exceeds the scope of this paper and is left for future work.

V. EXPERIMENTAL RESULTS

We evaluate our approach on two test cases using the before mentioned IVD architecture and application from Figs. 5 and 6 and a more complex, multi-purpose architecture and

TABLE III
COMPARISON BETWEEN [7] AND OUR ALGORITHM USING OUR MULTI-PURPOSE ARCHITECTURE AND IVD.

MP-Architecture	Valves	Required Pin-Count	Execution time
Previous research	66	25	45
Proposed method	66	14	70
IVD	Valves	Required Pin-Count	Execution time
Previous research	46	20	81
Proposed method	46	14	107

according application shown in Figs. 14 and 15. Using our own architecture from Fig. 14 we demonstrate the trade-off between the number of required control pins and the schedule length in Fig. 13. The figure shows the control count, starting at 31, which is the initial number of VGs created, on the Y-axis and the execution time on the X-axis. The progression shows that the algorithm introduces multiple constraints that effect the schedule length during pin-count reduction. Other combinations do not affect the schedule length, for reasons as shown in Examples 2 and 3. This is however only true for this particular order in which the combinations have been made. For example, the combinations that reduced the control count from 24 to 16 do not generally have no effect on the schedule length. Instead, the scheduling constraint introduced by the combinations before, create a scenario similar to Example 3 for the following combinations. Further combinations bring the control count to 14, which is the minimal number of control pins required to run the application as determined by the algorithm.

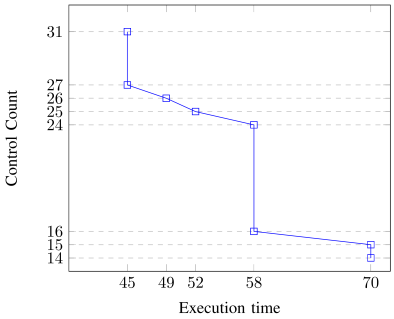


Fig. 13. Pin-count reduction chart

Tables III show a direct comparison between the pin-count reduction presented in [7] and our technique. The table contains the number of valves in each architecture, the number of controls required and the corresponding execution time for each method.

VI. CONCLUSIONS

In this paper we have proposed a new technique to reduce the required pin-count for flow-based biochips. Contrary to previous work, this method is able to trade off execution time to reduce the pin-count even further. Experimental results have shown that our algorithm is capable of reducing the pin-count significantly, while keeping the increase in schedule length acceptable. To produce realistic results, the current state-of-the-art routing model has been extended.

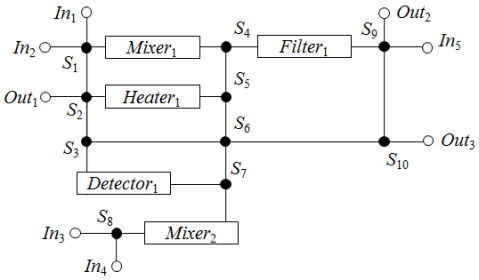


Fig. 14. Multi-purpose architecture able to transport fluid from every type of FFU to every other, allowing for a wide range of applications to be run.

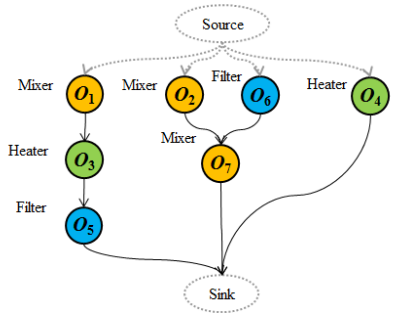


Fig. 15. One of many possible applications that can be carried out by the multi-purpose architecture.

REFERENCES

[1] H.-P. Chou, M. Unger, and S. Quake. A microfabricated rotary pump. *Biomedical Microdevices*, pages 323–330, 2001.

[2] T. A. Dinh, S. Yamashita, T.-Y. Ho, and Y. Hara-Azumi. A clique-based approach to find binding and scheduling result in flow-based microfluidic biochips. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 199–204, Jan 2013.

[3] M. Innovations. Assaymark. http://www.microfluidicinnovations.com/Datasheets/AssayMark_DS1301.pdf, 2013.

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

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

[6] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[7] W. Minhass, P. Pop, J. Madsen, and T.-Y. Ho. *Control Synthesis for the Flow-Based Microfluidic Large-Scale Integration Biochips*, pages 205–212, 2013.

[8] W. H. Minhass. System-level modeling and synthesis techniques for flow-based microfluidic very large scale integration biochips, 2012.

[9] W. H. Minhass, P. Pop, and J. Madsen. System-level modeling and synthesis of flow-based microfluidic biochips. In *Proceedings of Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 225–234, 2011.

[10] M. L. Raagaard and P. Pop. Pin count-aware biochemical application compilation for mvlsi biochips. *Design, Test, Integration and Packaging of MEMS/MOEMS*, pages 795–825, 2015.

[11] T. Thorsen, S. J. Maerkl, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.